

DIGITAL TRANSFORMATION IN ADVANCED MANUFACTURING

Big Data

The European Commission's support for the production of this publication does not constitute an endorsement of the contents, which reflect the views only of the authors, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Basic concepts of Big Data

Site: [DTAM Online Training Platform](#)
Course: Big Data
Book: Basic concepts of Big Data

Printed by: Vasileios Gkamas
Date: Friday, 27 October 2023, 4:18 PM

Description

In this section you will get introduced to what is Big Data, how it can be analysed, where we can see big data analysis in our surrounding world, which professional roles big data scientists can have and why Python is a good option for the Big Data domain.

Table of contents

1. What is Big Data
2. Different types of data
3. Big data benefits
4. Big data applications for advanced manufacturing
5. The data analysis process
6. Different roles in data science projects
7. Python for Big Data

1. What is Big Data

Defined by Gartner in 2001, big data is "high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation."

"Big Data is the Information asset characterized by such a High Volume, Velocity and Variety to require specific Technology and Analytical Methods for its transformation into Value." (De Mauro et al. 2016). The term "big data" refers to data that is so large, fast or complex that it's difficult or impossible to process using traditional methods. Wikipedia explains big data as the "extremely large data sets that may be analyzed computationally to reveal patterns, trends, and associations, especially relating to human behavior and interactions."

The act of accessing and storing large amounts of information for analytics has been around for a long time but their volume was not that big, or they were not that complex or were not produced at such a high speed. It's not possible to put a number on what quantifies big data, but it generally refers to figures around petabytes (1 petabyte = 1.000.000 GB) and exabytes (1 exabyte = 1.000.000.000 GB).

The amount of data we're creating continues to increase rapidly. Today, it is very easy to generate data whenever we go online, when we carry our GPS-equipped smartphones, when we communicate with our friends on social media, or even while we shop. In simple words, we can say that "every step we make is leaving a digital footprint".

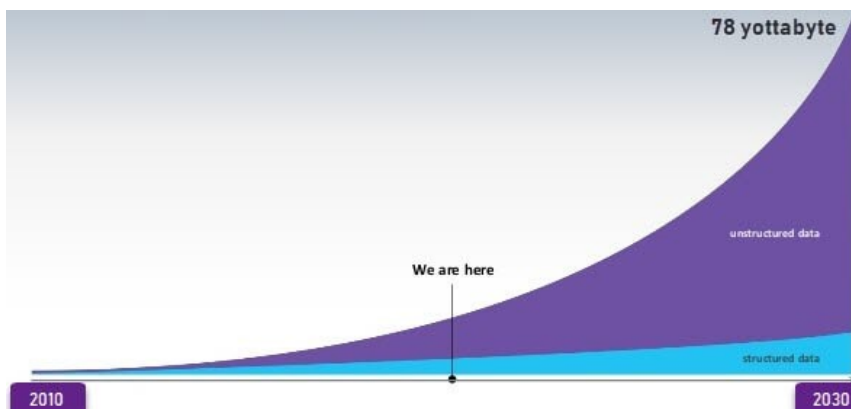


Figure 1. Data Growth over the years

The concept of big data gained momentum in the early 2000s when industry analyst Doug Laney defined big data as the 3 V's:

1. **Volume:** Refers to the amount of data and the form of data. Organizations collect data from a variety of sources, including business transactions, smart (IoT) devices, industrial equipment, videos, social media, and more.
2. **Velocity:** Refers to the time at which the data is collected and analyzed. With the growth in the Internet of Things, data streams into businesses at an unprecedented speed and must be handled in a timely manner. RFID tags, sensors, and smart meters are driving the need to deal with these torrents of data in near-real time.
3. **Variety:** Refers to the type of data collected. Data comes in all types of formats – from structured, numeric data in traditional databases to unstructured text documents, emails, videos, audio, stock ticker data, and financial transactions.

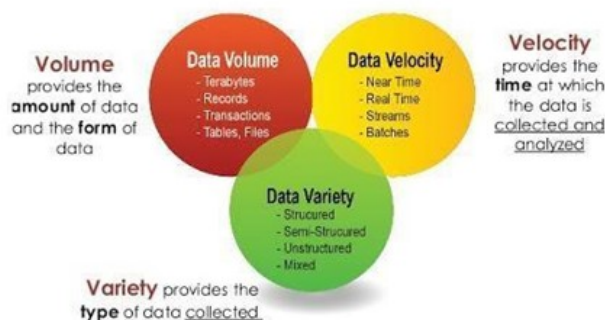


Figure 2. The 3 V's of Big Data

Now that we have discussed a bit about Volume, Variety, and Velocity we can understand the definition of Big Data as "high-volume, high-velocity and/or high-variety information assets that demand cost-effective, innovative forms of information processing that enable enhanced insight, decision making, and process automation."

Note that as years went by researchers have gradually added more Vs to Big Data. The fourth and fifth V's were Value and Veracity:

- **Value:** The primary interest in big data is probably for its business value. This is probably the most crucial characteristic of big data. Without value, there is no meaning of other big data characteristics.
- **Veracity:** Data veracity, in general, is how accurate or truthful a data set may be. In the context of big data, however, it's not just about the quality of the data itself but how trustworthy the data source, type, and processing of that data is.

Let's have a look at some additional V's that have been associated with Big Data (keep in mind that this list is not exhaustive and that there are different opinions on which and how many these V's are):

- **Viability:** It refers to carefully selecting those attributes in the data that are most likely to predict outcomes that matter most to organizations. As many big data scientists believe that 5% of the attributes in the data are responsible for 95% of the benefits, paying attention to the most important attributes can be very rewarding:
- **Validity:** Validity has some similarities with veracity. As the meaning of the word suggests, the validity of big data means how correct the data is for its purpose. Interestingly a considerable portion of big data remains useless, which is considered as 'dark data.' The remaining part of collected unstructured data is cleansed first for analysis.
- **Volatility:** Big data volatility refers to how long is data valid and how long should it be stored. In this world of real-time data, you need to determine at what point is data no longer relevant to the current analysis.

2. Different types of data

Big Data includes three types of data depending on its internal structure:

- **Structured:** data stored and organized in a fixed format, like for instance in a database.

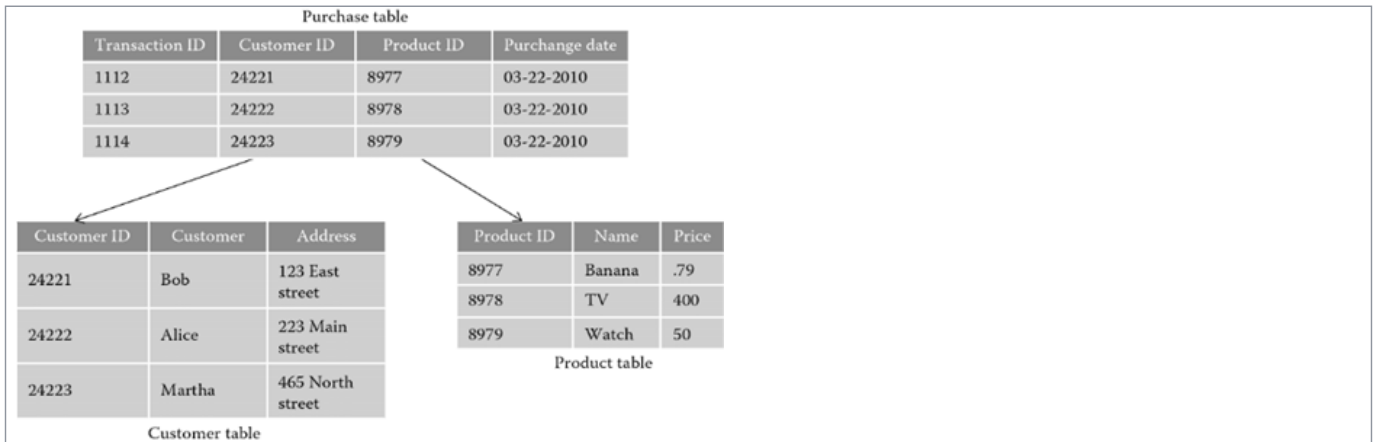


Figure 3. Tables in a relational database database

- **Unstructured:** Any data with unknown form or structure. A typical example of unstructured data is a heterogeneous data source containing a combination of simple text files, images, videos etc., like the page returned by 'Google Search' when we provide a search keyword.

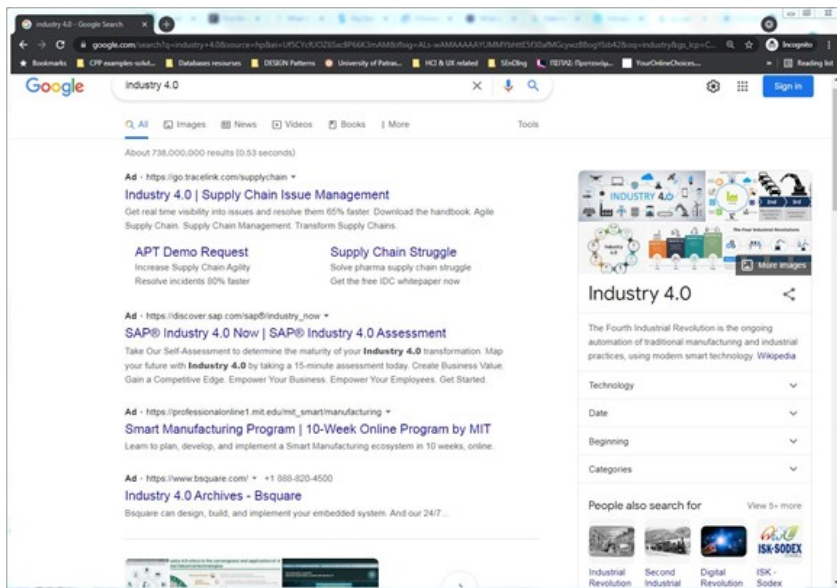


Figure 4. Google search results page when using "industry 4.0" as the search term

- **Semi-structured:** data that can't be organized in relational databases or doesn't have a strict structure, but it has some structural elements or a loose organization. Emails, for example, are semi-structured as they have some fixed fields like Sender, Recipient, Subject, Date, etc., but inside the body information is unstructured.

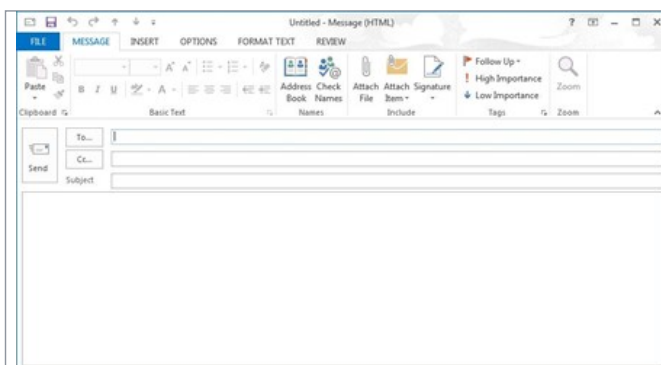


Figure 5. eMail fields

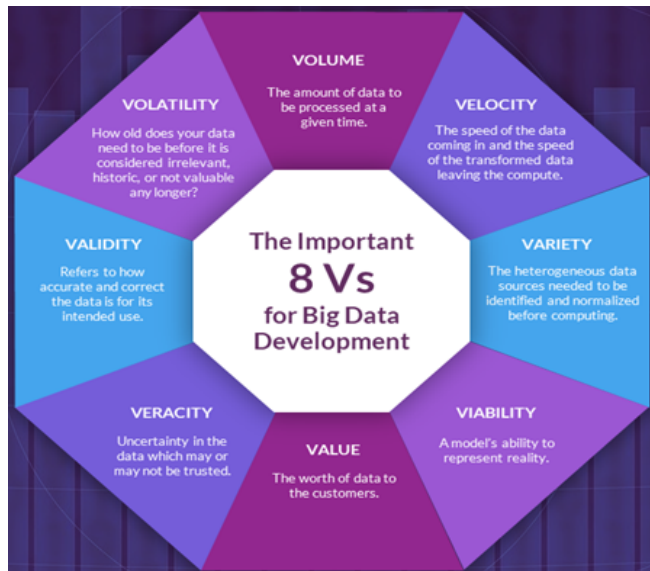


Figure 6. Additional Big Data V's

Big data can be used in almost every digitized industry – healthcare, financial, retail, and beyond. Let's have a look at some examples

- **Media and entertainment:** Analyzing big data allows us to generate more revenue and provide personalized experiences to audiences. For example, companies like Hulu and Netflix work with a variety of big data daily to analyze user tendencies, preferred content, trends in consumption, and much more. Also, think about all those ads targeted more strategically thanks to big data analytics software, that allow companies to understand the types of their customers and select the correct ads to display.
- **Finance:** Big data has fundamentally changed the finance industry, particularly stock trading. The introduction of quantitative analysis models has marked a shift from manual trading to trading supported by technology. These models analyze big data to predict outcomes of certain events in the financial world, make accurate trade decisions, minimize risk using machine learning, and even measure market sentiment using opinion mining.
- **Healthcare:** The healthcare industry is one of the largest recent adopters of big data analytics, where big data is collected and analysed in a patient-centric way. For example, there are companies that have managed to predict negative health events that seniors could experience from home-care thus reducing hospitalizations and ER visits among chronically ill patients. Also, we can use historical big data from healthcare providers to identify and analyze certain risk factors in patients. This is useful for earlier detection of diseases, allowing doctors and their patients to take action sooner. Last but not least, big data can identify disease trends based on demographics, geographics, socio-economics, and other factors.
- **Education:** Big data can help schools understand the unique needs of students by blending traditional learning environments with online environments. This allows educators to track the progress of their students and identify gaps in the learning process. Also, big data can be used to reduce dropout rates by identifying risk factors in students who are falling behind in their classes.
- **Retail:** Big data analytics allows retail companies to provide a variety of services and understand more about their customers. For instance, Amazon has set the golden standard when it comes to applying big data for product recommendations based on past searches on its platform. Using predictive analytics, Amazon and other retailers are able to accurately predict what you're likely to purchase next. Another example is Walmart that regularly analyzes changes in weather to see any patterns in product demand. Also, big data allows retail companies to handle cases that require fast action like product recalls (we can identify who purchased the product and reach them fast).
- **Manufacturing** is another industry with many Big Data applications, but this is something we will discuss in a next section!

3. Big data benefits

Big data allows us to effectively manipulate huge amounts of data, make sense of these data and make informed decisions. When used properly, big data can offer several crucial advantages like those listed below.

- **Better Decisions:** Big data offer companies greater probability of discovering patterns and insights in data which leads to data-driven insights which in turn provide a foundation for more informed and reliable decision-making.
- **Productivity and Efficiency:** with tools and techniques offered by big data organizations can process more information at faster speeds, which boosts personal productivity levels for individual employees and also the organization gains insights about its own operations and ways to become more productive.
- **Reduced Costs:** Streamlining operations, improving efficiency, and increasing productivity by using big data can introduce significant cost savings to an enterprise and increased overall profitability. By monitoring and analyzing internal operations organizations can better predict future demands in supply chain, maintenance scheduling and quality control.
- **Better Customer Service:** Technical support powered by big data, machine learning (ML), and artificial intelligence (AI) can greatly improve the standard of response and follow-up offered to consumers. Responsible use and analysis of customer and transaction data enable organizations to offer personalized support to customers, leading to greater engagement with brands and increased customer loyalty.
- **Fraud Detection:** In industries like financial services or healthcare, it's just as important to know what's going wrong as it is to know what's going right. AI and machine learning systems with big data can easily detect erroneous transactions, fraudulent activity indicators, and anomalies in collected and analyzed data and set off proper alarms.
- **Greater Agility:** Developments in the real-time processing of big data through stream analytics enable organizations to become more responsive and flexible in their internal operations (product development, innovation), and speed to market.

As you can understand, an organization has a lot to gain from using big data analytics in its day-to-day operation. To fully benefit from big data though, organizations need to hire data scientists and other big data professionals who know how to design, deploy, and manage infrastructure and achieve results from analytics. There's currently a serious skills shortage in these areas even though salaries are quite high. Apart from these high-level professionals an organization that decides to use big data will also need to train its employees so that they can efficiently operate in this new work environment. So, DTAM training is an excellent choice with great potential!

4. Big data applications for advanced manufacturing

Big data and the IoT can help manufacturing organizations in many ways. These technologies can help industries increase their automation level, which has the potential to reduce their production downtime and improve supply chain management. In a nutshell, big data and IoT can help industries track closely their operations and based on this close tracking, make better decisions and improve the way they operate.

Big Data Analytics and Maintenance: People that have been trained on Big Data can make a positive difference in the areas of maintenance and automation. For instance, they can mine performance data from machines to get insights about how to maintain equipment at its best level of performance. This exciting new area is called predictive maintenance. Being able to predict equipment failure has the potential to save manufacturing plants significant maintenance time and boost production rates. It also allows manufacturers to improve the quality of their products.

Supply Chain Improvements: Big data and the IoT also provide information about the supply chain. Specifically, these technologies can be used to monitor supply chain details in real-time and provide the information to industries and any connected suppliers. Among the details available to view are material flow and manufacturing cycle times of products. Using big data in this way can reduce product costs, as well as saving valuable time, all while allowing businesses to enjoy a bigger return on investment.

Data in Modern Manufacturing Processes: Currently, more and more industries use data to improve their operations. While process data was previously used primarily for tracking purposes, now it is showing businesses ways that they can improve existing processes. Trained analysts can spot patterns in data and draw insights from that information for manufacturers to act on as soon as possible. This is possible as more and more businesses integrate their existing systems with tools that allow them to communicate IoT devices. Collecting and analyzing data offers manufacturers valuable information about how their products perform.

So, Big Data and IoT are gradually becoming a 'hot' combination that can bring added value to industries. Just to have a general idea of what is yet to come, take a look at the graphic below from [a study by Deloitte](#) which shows the use of supply chain capabilities from Big Data currently in use and their expected use in the future.

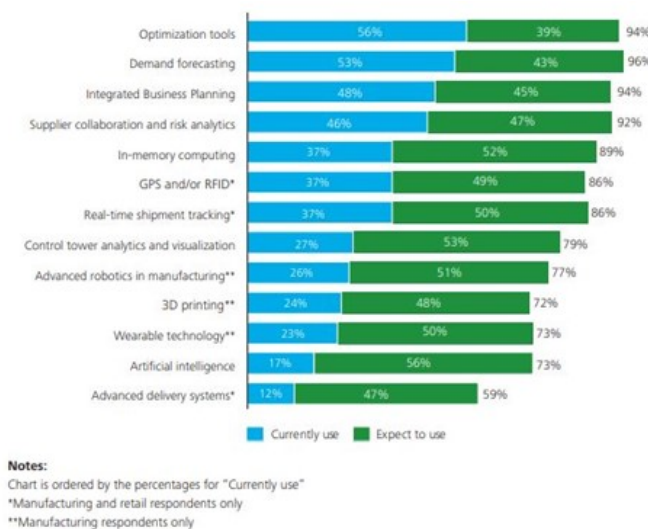


Figure 7. Use of supply chain capabilities (current and foreseen). Source: [Deloitte](#)

5. The data analysis process

Big data can be analyzed at various levels of complexity, and offer different levels of added value respectively:

- **Descriptive analytics** examines what happened in the past. Monthly revenue, quarterly sales, yearly website traffic, and so on. These types of findings allow an organization to see trends.
- **Diagnostic analytics** considers why something happened by comparing descriptive data sets to identify dependencies and patterns. This helps an organization determine the cause of a positive or negative outcome.
- **Predictive analytics** seeks to determine likely outcomes by detecting tendencies in descriptive and diagnostic analyses. This allows an organization to take proactive action—like reaching out to a customer who is unlikely to renew a contract, for example.
- **Prescriptive analytics** attempts to identify what business action to take. Even though this type of analysis can offer significant benefits (such as addressing potential problems or staying ahead of industry trends), it often requires the use of complex algorithms and advanced technology, such as machine learning.

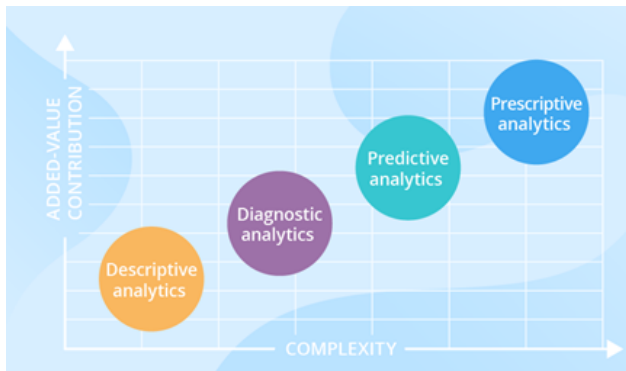


Figure 8. Added-value contribution vs complexity (Source: [ScienceSoft](#))

6. Different roles in data science projects

The Big Data domain is so wide that there are many specific roles (different job profiles). In fact, it is not surprising that big data skills is among the most in-demand skills of our century. In the following you can see the most well-known along with their job description.

Comparing big data roles



Figure 9. Roles in the Big Data domain (Source: whatis.techtarget.com)

Data scientists are widely perceived as fundamental in driving the world of big data forward. They collect, analyze, manage, structure and interpret large volumes of data from a range of sources. Data scientists then use reporting tools to pinpoint patterns, trends and interrelationships between the various data sets.

Data and computer engineers and architects are critical to supporting data scientists. They create the underpinning software architecture and design, build, and manage the infrastructure and scalable data management systems that data scientists need to perform their analysis.

Big data analysts detect and analyze actionable data, such as hidden trends and patterns. By fusing these findings with their in-depth knowledge of the market in which their organizations operate, they can help leaders formulate informed strategic business decisions.

Big data developers apply their deep understanding of technologies such as Hadoop and Apache Spark with programming languages such as Java, Python and Scala to process data. Using their functional programming fluency, they can effectively integrate data into broader big data platform ecosystems.

Big data specialists interrogate, ingest, analyze and transform complex sets of data. This ensures the necessary data is made available to the other team members who use it to uncover actionable insights and provide recommendations to improve business outcomes.

7. Python for Big Data



[source: eduinpro.com]

Choosing a programming language for the Big Data field depends highly on the specifics of the project and its goal. There is though one programming language that is a good choice regardless of the situation, and that is Python, a language that fits perfectly for Big Data projects because it is easily readable and offers full statistical analysis support. This way, big data experts can accomplish their goals with less coding and excellent library support. Some additional reasons why programmers choose Python for their big data tasks include:

- Both Python and Hadoop are open-source big data platforms, and that's why Python is securely more compatible with Hadoop than any other programming language. Developers prefer to use Python with Hadoop because of its **extensive support for libraries**.
- Python's **high speed for data processing** makes it optimal for usage with Big Data. Python codes are executed in a fraction of the time needed by other programming languages because of its simple syntax and easy-to-manage code.
- Big data analysis usually deals with complicated problems that need community support for solutions. Python has **large and active community support**, which helps data scientists and programmers with expert backing on coding-related issues. Also, corporate support is a significant part of the success of Python for Big Data. Top tech companies like Facebook, Instagram, Netflix, etc., use Python in their products.

In the charts below (from [JetBrains](https://jetbrains.com)) you can see the most popular programming languages for data analysis, data engineering, and machine learning (tasks that closely relate to big data). As you can Python is the most popular choice for all three categories.

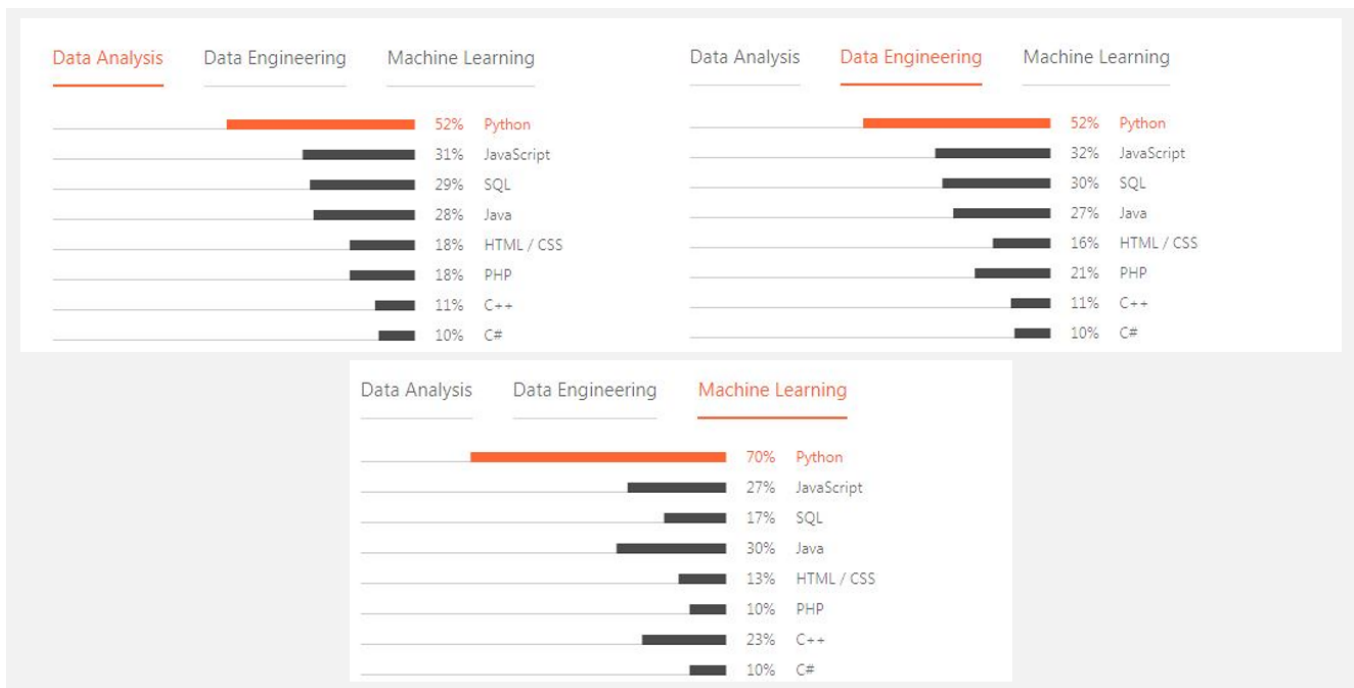


Figure 9. Primary language per type of task (source: [JetBrains](https://jetbrains.com))

Take a look at the diagram below. Python is also the primary language across all major business sectors including manufacturing, which is the focus of this training. No wonder we have chosen Python for our modules!

Core IT business	Banking / finance	Education and science	Sales / Distribution / Retail	Manufacturing	
46%	55%	66%	43%	47%	Python
33%	27%	23%	37%	32%	JavaScript
29%	34%	19%	21%	20%	Java
29%	36%	21%	40%	29%	SQL
20%	12%	15%	29%	15%	PHP
16%	15%	16%	22%	18%	HTML / CSS
14%	9%	6%	9%	10%	TypeScript
11%	11%	10%	9%	28%	C#
10%	7%	19%	6%	12%	C++

Figure 10. Primary language per sector (source: [JetBrains](#))

Introduction to Python for data analysis

Site: [DTAM Online Training Platform](#)
Course: Big Data
Book: Introduction to Python for data analysis

Printed by: Vasileios Gkamas
Date: Friday, 27 October 2023, 4:19 PM

Table of contents

1. Python installation
2. Using the Jupyter Notebook
3. Write a Python program using Anaconda Prompt or terminal
4. Downloading files with Python
5. Modules
6. Main libraries of Python for Data Analysis

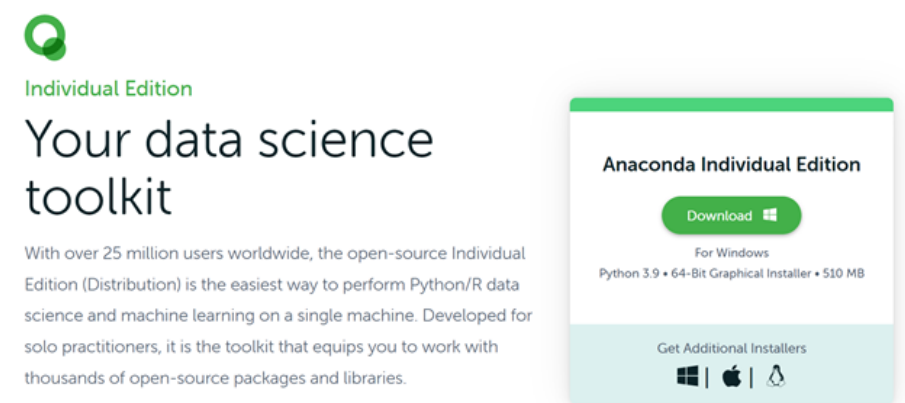
1. Python installation

In the [Python introduction](#) section we used the IDLE Shell to experiment with some simple Python code. For the purposes of Big Data analysis, we propose to use a more advanced programming environment that will give easy access to the required tools.

To this end, a good idea is to use the Anaconda distribution. It works similarly under all the major operating systems, and it includes many libraries that are essential for data analysis.

Anaconda installation

Just visit the Anaconda webpage: <https://www.anaconda.com/products/individual> and download the proper individual version for your operating system and its architecture.



Scrolling down this page you can see all available installers depending on your operating system:

Windows 	MacOS 	Linux 
Python 3.9	Python 3.9	Python 3.9
64-Bit Graphical Installer (510 MB)	64-Bit Graphical Installer (515 MB)	64-Bit (x86) Installer (581 MB)
32-Bit Graphical Installer (404 MB)	64-Bit Command Line Installer (508 MB)	64-Bit (Power8 and Power9) Installer (255 MB)
		64-Bit (AWS Graviton2 / ARM64) Installer (488 MB)
		64-bit (Linux on IBM Z & LinuxONE) Installer (242 MB)

Installation on Windows:

Execute the downloaded file and follow the instructions to install it in a directory on your computer. The installer will create a Start menu shortcut for Anaconda Navigator.

Installation on Linux:

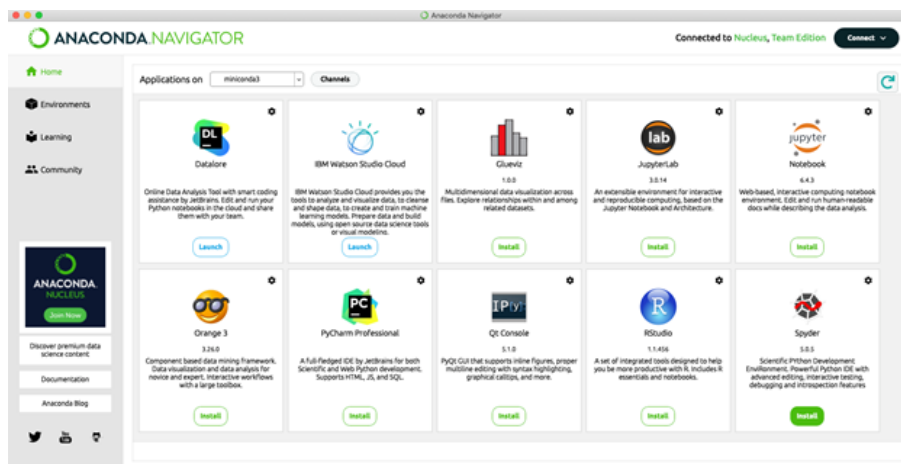
Locate the .sh installer file that you downloaded and run it with the command (adjust the name of the file accordingly):

```
bash ~/Downloads/Anaconda3-2020.02-Linux-x86_64.sh
```

Follow the instructions. Finally, to test if it has been installed correctly, open a terminal and run:

```
anaconda-navigator
```

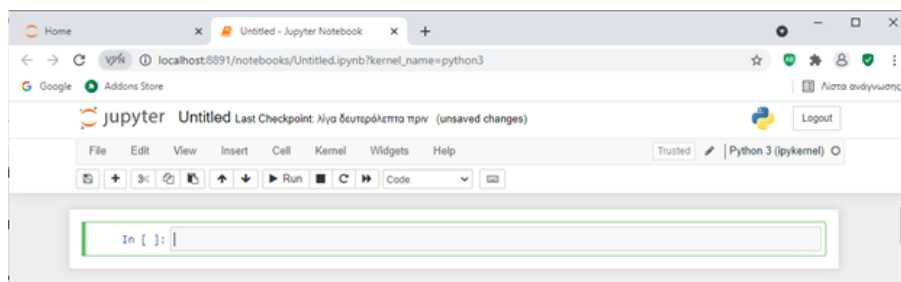
The Anaconda Navigator is a graphical interface for launching useful Python programs without having to use the command line.



How can you run code with Navigator? The simplest way is with Spyder. From the Navigator Home tab, click **Spyder**, and write and execute your code.

You can also use **Jupyter** Notebooks the same way. Jupyter Notebooks are an increasingly popular system that combine your code, descriptive text, output, images, and interactive interfaces into a single notebook file that is edited, viewed, and used in a web browser.

The Jupyter application is a notebook environment that runs in the web browser and allows running code interactively. It allows combining code execution with rich text for explanations and comments and using visualizations and media.



Anaconda also includes PyCharm (<https://www.jetbrains.com/pycharm/>), a popular IDE offered by JetBrains, that has a free community edition. PyCharm can be downloaded and installed independently as well (meaning that you do not have to install Anaconda first).



You can watch this short [video](#) (12 minutes) for a quick tour around the Anaconda work environment.

A short note about Anaconda and Python for Advanced Manufacturing Case Studies

According to anaconda.com "manufacturers use Anaconda and powerful open-source software for demand forecasting and supply chain optimization, predictive maintenance, root cause analysis, and quality control". They more specifically mention the following tasks as the most widely used by manufacturing when it comes to data science:

- building predictive analytics models to analyze historical sales data, weather data, economic patterns from various locations to forecast demand and optimize their supply chains,
- using image processing tools and anomaly detection to recognize and eliminate faulty parts before they go out to customers,
- combining sensors with image and audio processing tools to develop models that predict equipment degradation and prevent downtime.

2. Using the Jupyter Notebook

In the previous section, we installed **Anaconda** and used the **Jupyter Notebook** to write and execute a simple Python script. In this section we will focus on the syntax of the Python Programming Language. Before that, let's take a look on the *Jupyter Notebook* and some of its features.

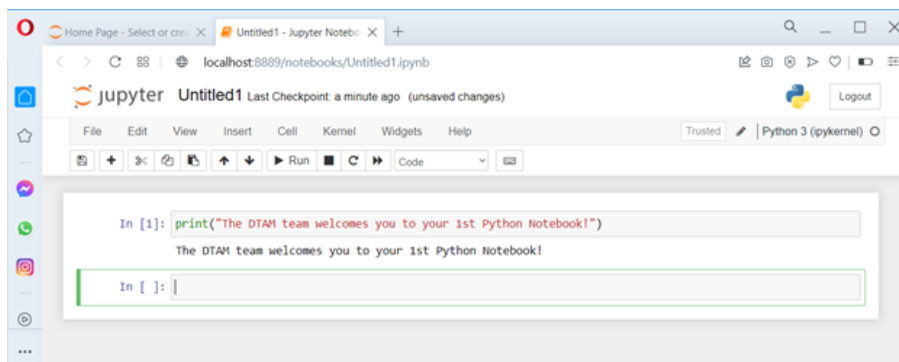


Figure 1: Running script in Jupyter Notebook

In *Jupyter* we can add multiple cells where we can write either our Python scripts or rich text with markup. We can even use HTML code and embed images, videos and other media. Using some plotting packages, we can dynamically generate visualizations from our python script. *Jupyter Notebook* files are stored with an ".ipynb" extension, they are not actual python files (".py" extension)

The idea behind *Jupyter* is that someone opening our Notebook could get both a rich document that describes our work with our results, as well as the scripts that we have used. They can run interactively these scripts and reach the same conclusions or even make modifications to them.

In *Jupyter* we can have more than one cells. We can click inside a cell to write content to it. We can press the 'Escape' key to leave the cell. We can use the Insert menu to add additional cells or the 'A' (insert Cell Above) and 'B' (insert Cell Below) shortcut keys. There is a dropdown field where we select the type of data of the cell. We can select whether it contains **Code** or **Markdown**.

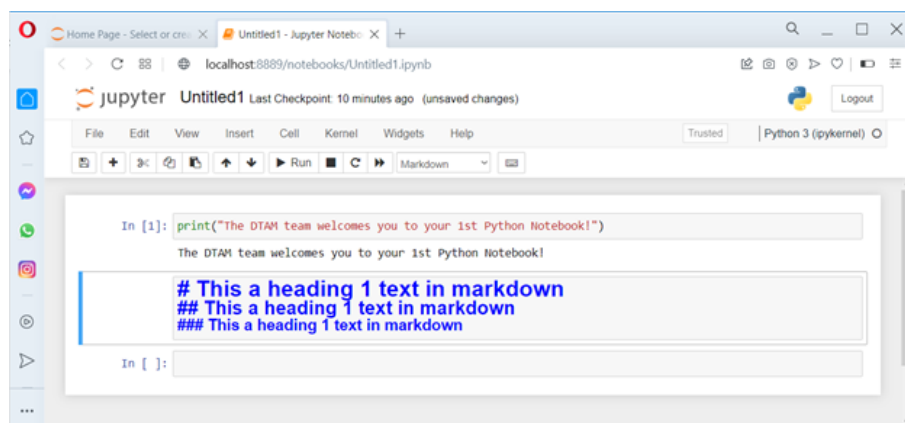


Figure 2: Code and Markdown Cells in Jupyter

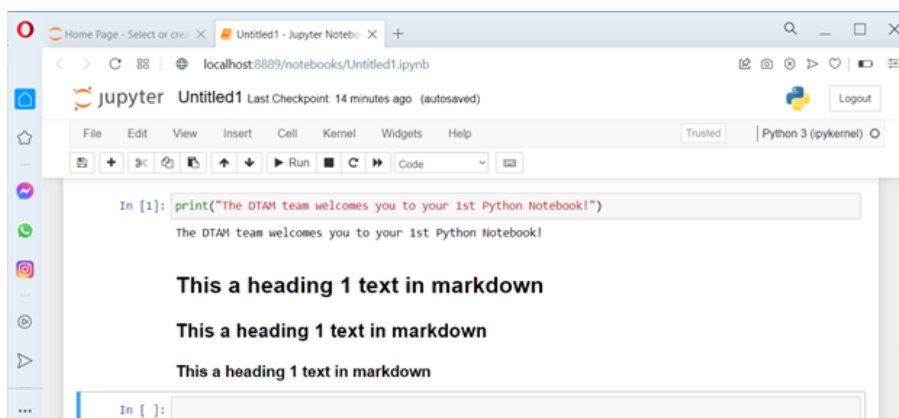


Figure 3: the formatted text after executing the markdown cell

For Markdown cells, we need to use specific syntax to annotate our text. We give some examples to get you started:

Style	Syntax	Example
Bold	** ** or __ __	**This is bold text**
Italic	* * or _ _	*This text is italicized*
Strikethrough	~~ ~~	~~This was mistaken text~~
Bold and nested italic	** ** and _ _	**This text is <i>_extremely_</i> important**
All bold and italic	*** ***	***All this text is important***

Links: To link to an external site surround the link with two underscores like bellow:

__[GitHub Pages] (<https://pages.github.com/>)__

Quoting: We use backticks to annotate quoted text. For example: `quoted text`. We can use triple backticks to quote multiple lines:

```quoted multiline text```

**Lists:** Use – or \* in front of each line. You can use numbers in front of each line for ordered lists. You can use Indentation for nested lists.

You can read more about markdown syntax here:

<https://help.github.com/en/articles/basic-writing-and-formatting-syntax>

We write our Python scripts (code) in Cells of “Code” and we can use the ‘Run’ button or ‘**Ctrl+Enter**’ to execute the code. We can also execute the code from the options of the ‘Cell’ Menu. We can choose to run specific cells or groups of them (Run All, Run All Below etc.)

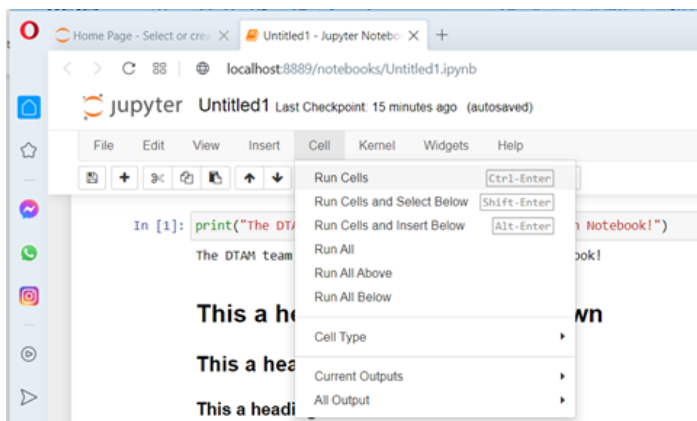


Figure 4: The Cell Menu in Jupyter

In Python scripts there are 3 ways of creating comments:

```
Simple Comment
"""This is a
multiline comment."""
'''This is also a
multiline comment.'''
```

The first way with # is used anywhere in the code and comments out the remaining text of that line.

The other methods allow quickly commenting out multiple lines of code and are mostly used at the start of a function to explain its usage.

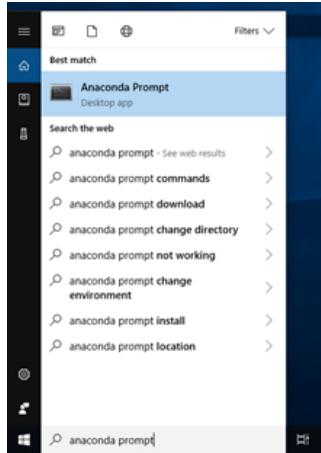


### 3. Write a Python program using Anaconda Prompt or terminal

The steps below are from the Anaconda User's Guide (you can visit <https://docs.anaconda.com/anaconda/user-guide/> for any questions you may have on using the Anaconda Navigator and its applications).

1. **Open Anaconda Prompt.** Choose the instructions for your operating system.

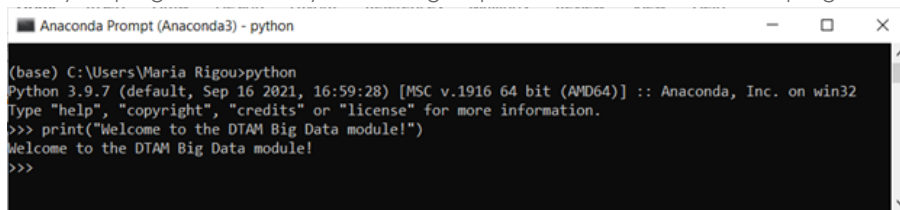
**Windows.** From the Start menu, search for and open "Anaconda Prompt":



**macOS.** Open Launchpad, then click the terminal icon.

**Linux.** Open a terminal window.

2. **Start Python.** At Anaconda Prompt (terminal on Linux or macOS), type `python` and press Enter. The `>>>` means you are in Python.
3. **Write a Python program.** At the `>>>`, type `print("Welcome to the DTAM Big Data module!")` and press Enter. When you press enter, your program runs and your message is printed to the screen. You're programming in Python!



4. **Exit Python.** On Windows, press CTRL-Z and press Enter. On macOS or Linux type `exit()` and press Enter.

## 4. Downloading files with Python

For working with files, Python offers the **open()** function. It returns a **file object** that we can use for reading/writing data. We need to specify a filename and a mode for opening it:

- "r" **Read:** Read content from a file that already exists(default mode)
- "a" **Append:** Open file to add content at the end. Creates the file if it does not exist
- "w" **Write:** Open a file to write content. Creates the file if it does not exist
- "x" **Create:** Create the file if it does not exist

We can add another character for the mode to specify:

- "t" **Text:** Handle the file as text
- "b" **Binary:** Handle the file as binary code

After opening a file with "r" mode, we can read data with the **read()** method. By default it returns the whole text of the file but we can also specify how many characters to read. Let's assume that we want to read my\_file.txt which is stored in C:\

```
f=open("my_file.txt","r")
```

```
print(f.read(10))# Print the first 10
characters of the file
```

The **readline()** method gets one line from the file. We can call it repeatedly to read each line from the file.

```
f=open("C:\my_file.txt","r")
print(f.readline())# Print the first Line
print(f.readline())# Print the second Line
```

We can use a simple **for** loop to iterate the lines of the file!

```
f=open("C:\my_file.txt","r")
for x in f: #For each line
in the file
 print(x) #Print it
```

We can read the lines of a file and store them as a list with the **list()** function or the **readlines()** method.

```
f=open("C:\my_file.txt","r")
list(f)
```

```
f=open("C:\my_file.txt","r")
```

```
f.readlines()
```

We need to close the file after we are done with it. The method to do this is **close()**.

```
f=open("C:\my_file.txt","r")
print(f.read())
f.close()
```

When we open a file to write content, we need to specify if we want to append to the end of it ("a") or overwrite the existing content ("w"). We use the **write()** method to write to the file. If the value we want to write is not a string we need to convert it with the **str()** function.

```
Open my_file.txt stored
in C:\ and add something at the end
f=open("C:\my_file.txt","a")
f.write("This Text will be
added at the end!")
f.close()
Open file and write
something (the old content is lost)
f=open("C:\my_file.txt","w")
f.write("This is the new
content of the file!")
f.close()
```

An overview of our options to create a new file:

Mode	File Exists	File does not exist
Create "x"	Returns Error	Create a file
Append "a"	Will Write at the End	Create a file
Write "w"	Will Overwrite Content	Create a file

To delete a file, there is a **remove()** function in the **os** module (we need to import it first).

```
import os
os.remove("C:\my_file.txt")
```

Trying to delete a file that does not exist will produce an error. We can check if the file exists with the **os.path.exists()** method.

```
import os
if os.path.exists("C:\my_file.txt"):
 os.remove("C:\my_file.txt")
else:
 print("file not found")
The os module, offers the
mkdir() function to create a folder:
import os
os.mkdir("my_folder")
```

The **os** module, offers the **listdir()** function to get a list of the files inside a folder. The following script iterates over the files of a directory, and prints their names:

```
import os
my_files = os.listdir('my_folder')
for filename in my_files:
 print(filename)
```

We can also use the **glob()** method from the Path class in the pathlib module. This allows us to specify what files to retrieve

```
from pathlib import Path
path = Path('my_folder')
for x in path.glob('*.'):
 print(x)
The os module, offers the rmdir()
function to delete a folder:
import os
os.rmdir("my_folder")
```

An Alternative method to open and use files in Python is to use the **with** statement. The with statement simplifies exception handling by encapsulating common preparation and cleanup tasks. It will automatically close the file, even if an exception was raised at some point! It is considered good practice to open files with the **with** statement.

```
READ FROM
FILE
with open("testfile.txt", 'r') as f:
 data = f.read()
 ... code ...
WRITE TO FILE
with open("testfile.txt", 'w') as f:
 f.write("test123\n")
 ... code ...
```

## 5. Modules

**Modules** are files (with “.py” extension) containing a set of functions that you can include in your application. A **Package** is a collection of Modules. We have already seen how to import a module (or package) in our code:

```
#Import the math module
import math
print(math.pi)

#Import only pi
from math import pi
print(pi)

#Import everything from math
from math import*
print(pi)
print(factorial(6))
```

Each module has a variety of functions inside it. Here are some examples from the **math** module we have already used:

```
x = math.sqrt(16) #square root (x=4)
x = math.pow(2,3) #equivalent to 23 (x=8)
x = math.ceil(4.3) #rounds up (x=5)
x = math.floor(7.8) #rounds down (x=7)
x = math.log(34) #loge34 (x=3.52)
x = math.log10(100) #log10100 (x=2)
x = math.sin(3.14) #angle in radians
x = math.cos(math.pi)
x = math.tan(math.pi* 2)
x = math.asin(-1) #inverse sine
x = math.sinh(1) #hyperbolic sine
```

Some other modules from the **Python Standard Library** that are often used:

- random
- time
- os (operating system)
- re (regular expressions)
- csv
- string
- copy
- gzip

Some **third party packages**, commonly used in data science:

- NumPy
- Pandas
- Scikit-Learn
- Matplotlib

We will talk about numpy, pandas and matplotlib in the next sections of the Big Data course. These need to be installed before you can import them. If you installed the **Anaconda** version, they should be already installed.

We can create our own modules (.py files) and import them in other python files to access the functions, classes and variables in them.

```
we save this code as a file with name "person.py"
class Person:
 def __init__(self, surname, age):
 self.surname= surname
 self.age= age

class Student(Person):
 def __init__(self, surname, age, year):
 super().__init__(surname, age)
 self.year= year

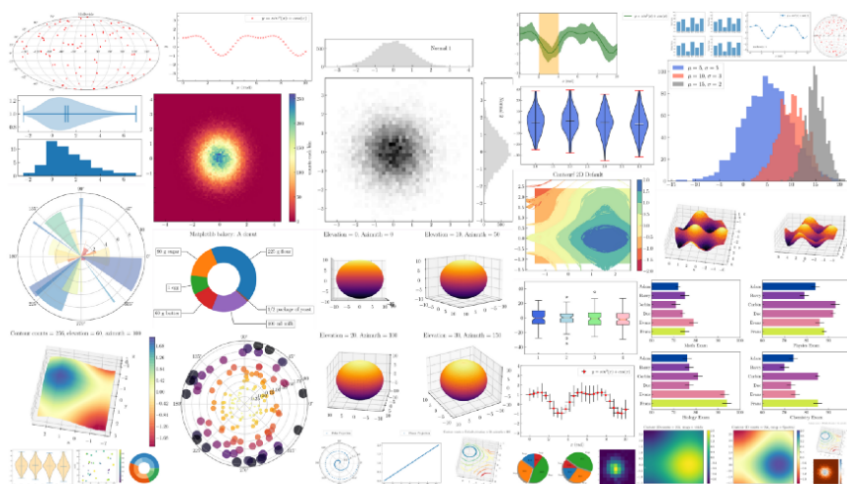
in another file / notepad we access the module
import person
betty =person.Student("Betty",28,2014)
print(betty.surname,betty.year)
#Output: Betty 2014
```

**Packages** are just collections of multiple modules. To make a new package, you create a new folder and then add inside a special file `__init__.py` that tells Python it is a package.

## 6. Main libraries of Python for Data Analysis

In the previous section we referred to the NumPy, Pandas, and Matplotlib libraries. Let's have a quick view of what each library offers before moving on to the next chapters where we discuss them in detail and show some practical examples of using them.

- **NumPy (Numerical Python)** is an open-source Python library that's used in almost every field of science and engineering. It's the universal standard for working with numerical data in Python. In Python we have lists that serve the purpose of arrays, but they are slow to process. NumPy aims to provide an array object that is up to 50x faster than traditional Python lists. This array object in NumPy is called `ndarray`, and it provides a lot of supporting functions that make working with `ndarray` very easy. Arrays are very frequently used in data science, where speed and resources are very important. NumPy is used by everyone, from beginning coders to experienced researchers doing state-of-the-art scientific and industrial research and development.
- **Pandas** is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. The name "Pandas" has a reference to both "**Panel Data**", and "**Python Data Analysis**". Pandas allows us to analyze big data and make conclusions based on statistical theories. Pandas can clean messy data sets, and make them readable and more understandable. Pandas gives you answers about the data. Like: Is there a correlation between two or more columns? What is average value? Max or max value? Pandas are also able to delete rows that are not relevant, or contain wrong values, like empty or NULL values. This is called *cleaning* the data.
- Visualization plays a huge role in bringing your data to life and uncovering the story it's trying to tell. The core package used for data visualization is **Matplotlib**. Matplotlib is an open-source graph plotting library in Python. We use it for creating static, animated, and interactive visualizations. As many people say, Matplotlib makes easy things easy and hard things possible. Below you can see some examples of visualizations we can produce with Matplotlib [\[source\]](#).





# The NumPy library in Python for the creation of large, multi-dimensional arrays and matrices

Site: [DTAM Online Training Platform](#)

Course: Big Data

Book: The NumPy library in Python for the creation of large, multi-dimensional arrays and matrices

Printed by: Vasileios Gkamas

Date: Friday, 27 October 2023, 4:19 PM

## Table of contents

1. The NumPy Array
2. The ndarray Class
3. Array Creation
4. Printing Arrays
5. Indexing, Slicing and Iterating
6. Shape Manipulation

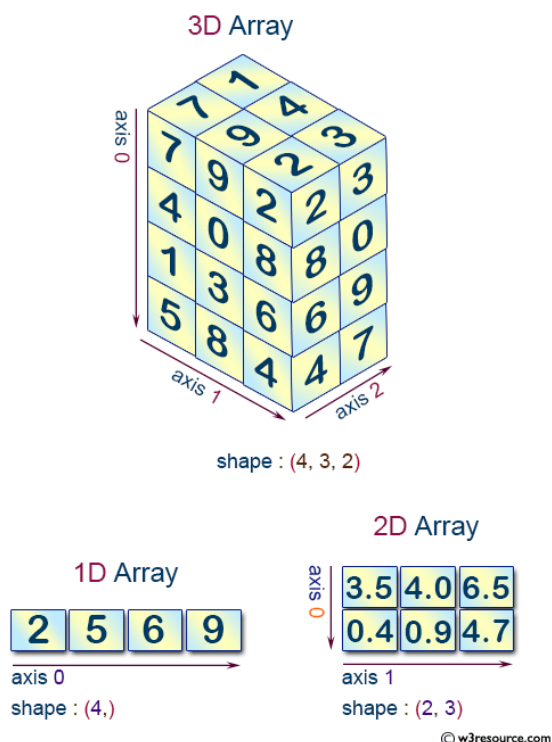
# 1. The NumPy Array

NumPy contains among other things, the powerful high-performance multi-dimensional array object, the **NumPy Array**, and the necessary tools for working with it. It is particularly useful for linear algebra and provides random number capabilities.

The NumPy Array is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. In NumPy dimensions are called axes. For example:

- a point in 3D space `[1, 2, 1]` has one axis. That axis has 3 elements in it, so we say it has a length of 3.
- for `[[ 1., 0., 0.], [ 0., 1., 2.]]`, the array has 2 axes. The first axis has a length of 2, the second axis has a length of 3.

Bellow you can see a nice graphical representation of the meaning of axes 0, 1 and 2 (source: [w3resource.com](http://w3resource.com))



## 2. The ndarray Class

NumPy's array class is called **ndarray**. It is also known by the alias `array`. Note that **numpy.array** is not the same as the Standard Python Library class **array.array**, that can only handle one-dimensional arrays.

The most important attributes of a **ndarray** object are shown in the following table:

Attribute	Description
<code>ndarray.ndim</code>	the number of axes (dimensions) of the array.
<code>ndarray.shape</code>	the dimensions of the array (Tuple of integers indicating the size of the array in each dimension). For a matrix with n rows and m columns, shape will be (n,m).
<code>ndarray.size</code>	the total number of elements of the array.
<code>ndarray.dtype</code>	an object describing the type of the elements in the array.  Additionally to the standard Python types, NumPy provides types of its own such as: <code>numpy.int32</code> , <code>numpy.int16</code> , and <code>numpy.float64</code>
<code>ndarray.itemsize</code>	the size in bytes of each element of the array.

Bellow you can see some simple Python examples of these attributes.

```
import numpy as np

a = np.array([[0, 1, 2, 3],
 [4, 5, 6, 7],
 [8, 9, 10, 11]])

print(a.shape) # (3, 4)
print(a.ndim) # 2
print(a.size) # 12
print(a.dtype.name) # int32
print(a.itemsize) # 4
print(type(a)) # <class 'numpy.ndarray'>
```

### 3. Array Creation

You can create an array from a regular Python list or tuple using the **array** function. The type of the resulting array is deduced from the type of the elements in the sequences.

```
import numpy as np

a = np.array([[0, 1, 2, 3],
 [4, 5, 6, 7],
 [8, 9, 10, 11]])

print(a.shape) # (3, 4)
print(a.ndim) # 2
print(a.size) # 12
print(a.dtype.name) # int32
print(a.itemsize) # 4
print(type(a)) # <class 'numpy.ndarray'>
```

You can also create arrays with these useful functions:

- **np.zeros**: Create an array of all zeros
- **np.ones**: Create an array of all ones
- **np.full**: Create a constant array
- **np.eye**: Create an Identity matrix
- **np.empty**: Create an array without initializing entries.
- **np.arange**: This function returns an array containing evenly spaced values within a given range (from start to stop). We must define start, stop and step values.
- **np.random**: An array filled with random values

Below you see some examples of using these functions:

```
import numpy as np

a = np.zeros((2,2))
print(a)
[[0. 0.]
[0. 0.]]

a2 = np.zeros(10, dtype=int)
print(a2)
[0 0 0 0 0 0 0 0 0 0]

b = np.ones((1,2))
print(b)
[[1. 1.]]

c = np.full((2,2), 7)
print(c)
[[7. 7.]
[7. 7.]]

d = np.eye(2)
print(d)
[[1. 0.]
[0. 1.]]
e = np.random.random((2,2))
print(e)
Example Output:
[[0.91940167 0.08143941]
[0.68744134 0.87236687]]
2x2 array of normally distributed random values with mean 0 and standard
deviation 1
```

```
e1 = np.random.normal(0, 1, (2, 2))
print(e1)
Example Output:
[[2.05102518 0.11751838]
[2.05363272 0.60144205]]
2x2 array of random integers in the interval [0, 10)

e2 = np.random.randint(0, 10, (2, 2))
print(e2)
Example Output:
[[7 3]
[0 7]]
```

## 4. Printing Arrays

When using print to display an array, the last axis is printed from left to right. The second-to-last is printed from top to bottom. The rest are also printed from top to bottom, with each slice separated from the next by an empty line. Based on that, One-dimensional arrays are printed as rows, Bidimensional arrays as matrices and Tridimensional arrays as lists of matrices.

```
a = np.arange(16)
print(a)
[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15]

b = np.arange(16).reshape(2,8)
print(b)
#[[0 1 2 3 4 5 6 7]
[8 9 10 11 12 13 14 15]]

c = np.arange(16).reshape(2,2,4)
print(c)
#[[[0 1 2 3]
[4 5 6 7]]
#
[[8 9 10 11]
[12 13 14 15]]]
```

## 5. Indexing, Slicing and Iterating

### Slicing

One-dimensional arrays can be indexed, sliced and iterated over, much like lists and other Python sequences. Multidimensional arrays can have one index per axis. These indices are given in a tuple separated by commas. When fewer indices are provided than the number of axes, the missing indices are considered complete slices:

`b[-1]` is equivalent to `b[-1,:]`

```
import numpy as np

Create the following rank 2 array with shape (3, 4)
[[1 2 3 4]
[5 6 7 8]
[9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

Use slicing to pull out the subarray consisting of the first 2 rows
and columns 1 and 2; b is the following array of shape (2, 2):
[[2 3]
[6 7]]
b = a[:2, 1:3]

A slice of an array is a view into the same data, so modifying it
will modify the original array.
```

The dots (...) represent as many colons as needed to produce a complete indexing tuple. For example, if `x` is an array with 5 axes, then:

```
x[1,2,...] is equivalent to x[1,2,:,:,]
x[...3] is equivalent to x[:, :, :, 3]
x[4,...,5,:] is equivalent to x[4, :, 5, :,]
```

### Integer array indexing

When you index into NumPy arrays using slicing, the resulting array view will always be a sub-array of the original array. In contrast, integer array indexing allows you to construct arbitrary arrays using the data from another array.

```
a = np.array([[1,2], [3, 4], [5, 6]])

An example of integer array indexing. The array will have shape (3,)
print(a[[0, 1, 2], [0, 1, 0]]) # Prints "[1 4 5]"

The above example is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]])) # Prints "[1 4 5]"

When using integer array indexing, you can reuse the same element
from the source array:
print(a[[0, 0], [1, 1]]) # Prints "[2 2]"

The above example is equivalent to this:
print(np.array([a[0, 1], a[0, 1]])) # Prints "[2 2]"
```

### Boolean array indexing



Boolean array indexing lets you pick out arbitrary elements of an array. Frequently this type of indexing is used to select the elements of an array that satisfy some condition.

```
a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2) # Find the elements of a that are bigger than 2
 # Returns a NumPy array of Booleans of the same shape as a, where
 # each element is true if that element of a is > 2.

print(bool_idx) # Prints:
 # [[False False]
 # [True True]
 # [True True]]

We use Boolean Array Indexing to construct an array consisting of
the
elements of 'a' that correspond to the True values of bool_idx
print(a[bool_idx]) # Prints "[3 4 5 6]"

We can do all of the above in a single concise statement:
print(a[a > 2]) # Prints "[3 4 5 6]"
```

## Iterating

**Iterating** over multidimensional arrays is done with respect to the first axis. However, if you want to perform an operation on each element in the array, you can use the **flat** attribute which is an **iterator** over all the elements of the array:

```
b = np.arange(6).reshape(3,2)
```

```
for row in b:
 print(row)
```

0	1
2	3
4	5

```
for element in b.flat:
 print(element)
```

0
1
2
3
4
5

## 6. Shape Manipulation

Next, we are going to see three common manipulations:

- Changing the shape of an array
- Stacking together different arrays
- Splitting one array into several smaller ones

An array has a shape given by the number of elements along each axis. The shape of an array can be changed with various methods.

The **reshape** method returns its argument with a modified shape, whereas the **resize** method modifies the array itself:

`a.reshape(2,3)`                      *Returns new Array*

`a.resize(2,3)`                      *Modifies the Array*

If a dimension is given as -1 in a reshaping operation, the other dimensions are automatically calculated:

`a.reshape(3,-1)`

```
a = np.arange(6).reshape(3,2)
print (a)
[[0 1]
[2 3]
[4 5]]
print (a.shape) # (3, 2)

print (a.ravel()) # returns the array, flattened
[0 1 2 3 4 5]

print (a.reshape(2,3)) # returns the array with a modified
shape
[[0 1 2]
[3 4 5]]

print (a.T) # returns the array, transposed
[[0 2 4]
[1 3 5]]
print (a.T.shape) # (2, 3)

print (a.shape) # (3, 2)
```

The function **column\_stack** stacks 1D arrays as columns into a 2D array. It is equivalent to **hstack** only for 2D arrays. The function **row\_stack** is equivalent to **vstack** for any input arrays.

For arrays with more than two dimensions:

- **hstack** stacks along their second axes
- **vstack** stacks along their first axes
- **concatenate** allows for an optional argument with the number of the axis along which the concatenation should happen

Several arrays can be stacked together along different axes:

```

a = np.arange(4).reshape(2,2)
print (a)
[[0 1]
[2 3]]

b = np.arange(4).reshape(2,2) + 4
print (b)
[[4 5]
[6 7]]
print (np.vstack((a,b)))
[[0 1]
[2 3]
[4 5]
[6 7]]
print (np.hstack((a,b)))
[[0 1 4 5]
[2 3 6 7]]
from numpy import newaxis

a = np.arange(4) # [0 1 2 3]
b = np.arange(4)+ 4 # [4 5 6 7]

print (np.column_stack((a,b)))
[[0 4]
[1 5]
[2 6]
[3 7]]

print (np.hstack((a,b))) # the result is different
[0 1 2 3 4 5 6 7]

print (a[:,newaxis]) # this allows to have a 2D columns vector
[[0]
[1]
[2]
[3]]

print(np.column_stack((a[:,newaxis],b[:,newaxis])))
[[0 4]
[1 5]
[2 6]
[3 7]]

print(np.hstack((a[:,newaxis],b[:,newaxis])))
the result is the same
[[0 4]
[1 5]
[2 6]
[3 7]]

```

Using **hsplit**, you can split an array along its horizontal axis, either by specifying the number of equally shaped arrays to return, or by specifying the columns after which the division should occur:

```
a = np.arange(12).reshape(3,4)
print(a)
```

0	1	2	3
4	5	6	7
8	9	10	11

```
for i in np.hsplit(a,2):
 print ("Array:")
 print (i)
```

0	1
4	5
8	9

2	3
6	7
10	11

```
for i in np.hsplit(a,(1,3)):
 # Split after 1st and 3rd column
 print ("Array:")
 print (i)
```

0
4
8

1	2
5	6
9	10

3
7
11

# The Pandas library in Python for data analysis

Site: [DTAM Online Training Platform](#)

Course: Big Data

Book: The Pandas library in Python for data analysis

Printed by: Vasileios Gkamas

Date: Friday, 27 October 2023, 4:19 PM

# Table of contents

1. Main data structures in Pandas
2. Data Frame
3. Combine Data Frames
4. Rows and Columns Selection
5. Sorting
6. Descriptive Statistics
7. Group By
8. File I/O
9. Missing values and duplicates
10. Other data cleaning tasks
11. Importing and manipulating large datasets

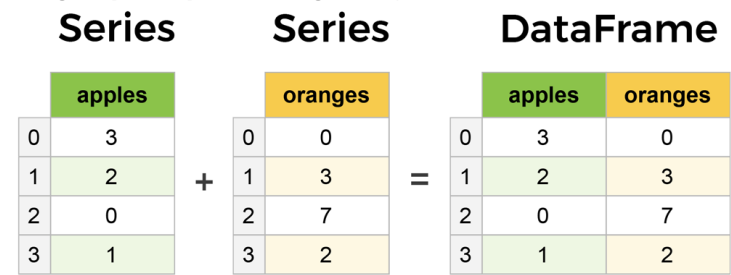
# 1. Main data structures in Pandas

Pandas is a library that provides high-performance, flexible **data structures** and tools for **data analysis**. It offers an **easy and intuitive** way to work with “relational” or “labeled” data. It is one of the most powerful **data manipulation** tools available in any language.

The main data structures in Pandas are:

- **Series**: One-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index.
- **DataFrame**: A 2-dimensional labeled data structure with columns that can each have a different type. It is generally the most commonly used pandas object.

The figure [\[source\]](#) below is a good representation of what each structure contains:



## 2. Data Frame

The **DataFrame** is a 2-dimensional labeled data structure. You can think of it like a spreadsheet or a database table. DataFrames are essentially multidimensional arrays with attached row and column labels, and often with heterogeneous types of data that in some cases may even be missing data!

### DataFrame Attributes:

- **dtypes**: List the types of the columns.
- **columns**: List the column names.
- **axes**: List the row labels and column names.
- **ndim**: Number of dimensions.
- **size**: Number of elements.
- **shape**: Return a tuple representing the dimensionality.
- **values**: NumPy representation of the data.
- **index**: The labels.

### DataFrame Methods:

- **head([n]), tail([n])**: Returns the first / last n rows.
- **sample([n])**: Returns a random sample of the data frame.
- **describe()**: Generates descriptive statistics.
- **max(), min()**: Returns max/min values for numeric columns.
- **mean(), median()**: Returns mean values for all numeric columns.
- **std()**: Calculates the standard deviation.
- **dropna()**: Drops all records with missing values.

Below you can see examples of creating two simple dataframes in Jupyter

The screenshot shows a Jupyter Notebook interface with the title "testingDataframe (unsaved changes)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a status bar (Trusted, Python 3 (ipykernel)), and a toolbar with icons for file operations, running code, and viewing output.

**Cell 3:** The code imports pandas as pd, defines a list of columns, and creates a DataFrame with two rows of data. The output shows the DataFrame with columns 'name', 'age', 'gender', and 'city'.

```
In [3]: import pandas as pd
columns = ['name', 'age', 'gender', 'city']
persons1 = pd.DataFrame([['Maria', 24, "F", "Patras"],
 ['Christine', 26, "F", "Athens"]],
 columns=columns)
print(persons1)
```

	name	age	gender	city
0	Maria	24	F	Patras
1	Christine	26	F	Athens

**Cell 4:** The code creates a DataFrame using a dictionary with two rows of data. The output shows the DataFrame with columns 'name', 'age', 'gender', and 'job'.

```
In [4]: persons2= pd.DataFrame(dict(name=['Vassilis', 'Betty'],
 age=[27, 22],
 gender=['M', 'F'],
 job=['Arta', 'Sparta']))
print(persons2)
```

	name	age	gender	job
0	Vassilis	27	M	Arta
1	Betty	22	F	Sparta



### 3. Combine Data Frames

We can concatenate DataFrames with the **append** method or the **concat** function:

```
print(persons1.append(persons2))
```

```
print(pd.concat([persons1, persons2]))
```

Also, we can join DataFrames with the **merge** function by specifying **on** which column and **how** to join them ('left', 'right', 'outer', 'inner').

Below we give an example that continues with the two DataFrames we created in the previous section (persons1 and persons2):

```
In [16]: persons=persons1.append(persons2)
```

```
In [17]: print(persons)
```

	name	age	gender	city	job
0	Maria	24	F	Patras	NaN
1	Christine	26	F	Athens	NaN
0	Vassilis	27	M	NaN	Arta
1	Betty	22	F	NaN	Sparta

```
In [18]: persons3 = pd.DataFrame(dict(
 name=['Maria', 'Christine', 'Vassilis', 'Kostas'],
 height=[171, 165, 190, 181]))
print(persons3)
```

	name	height
0	Maria	171
1	Christine	165
2	Vassilis	190
3	Kostas	181

```
In [19]: print (pd.merge(persons, persons3, on="name"))
```

	name	age	gender	city	job	height
0	Maria	24	F	Patras	NaN	171
1	Christine	26	F	Athens	NaN	165
2	Vassilis	27	M	NaN	Arta	190

```
In [20]: print (pd.merge(persons, persons3, on="name", how='left'))
```

	name	age	gender	city	job	height
0	Maria	24	F	Patras	NaN	171.0
1	Christine	26	F	Athens	NaN	165.0
2	Vassilis	27	M	NaN	Arta	190.0
3	Betty	22	F	NaN	Sparta	NaN


```
In [21]: print (pd.merge(persons, persons3, on="name", how='outer'))
```

	name	age	gender	city	job	height
0	Maria	24.0	F	Patras	NaN	171.0
1	Christine	26.0	F	Athens	NaN	165.0
2	Vassilis	27.0	M	NaN	Arta	190.0
3	Betty	22.0	F	NaN	Sparta	NaN
4	Kostas	NaN	NaN	NaN	NaN	181.0

#### Reshaping by pivoting

The **melt** function "unpivots" a DataFrame from wide format to long (stacked) format. The **pivot** method "pivots" a DataFrame from long (stacked) format to wide format.

In case you are wondering, a *pivot table* is a table of statistics that summarizes the data of a more extensive table. In practical terms, a pivot table calculates a statistic on a breakdown of values. For the first column, it displays values as rows and for the second column as columns (you can see an explanation [here](#)).

jupyter testingDataframe (unsaved changes)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run Code

```
In [22]: staked = pd.melt(persons, id_vars="name",
 |> var_name="variable",
 |> value_name="value")
 |> print(staked)
```

	name	variable	value
0	Maria	age	24
1	Christine	age	26
2	Vassilis	age	27
3	Betty	age	22
4	Maria	gender	F
5	Christine	gender	F
6	Vassilis	gender	M
7	Betty	gender	F
8	Maria	city	Patras
9	Christine	city	Athens
10	Vassilis	city	NaN
11	Betty	city	NaN
12	Maria	job	NaN
13	Christine	job	NaN
14	Vassilis	job	Arta
15	Betty	job	Sparta

```
In [23]: print(staked.pivot(index='name', columns='variable', values='value'))
```

	variable	age	city	gender	job
name					
Betty	22	NaN	F	Sparta	
Christine	26	Athens	F	NaN	
Maria	24	Patras	F	NaN	
Vassilis	27	NaN	M	Arta	

```
In []: |
```

## 4. Rows and Columns Selection

We can select specific columns using their names. Below you can see some examples of how things work.

The Jupyter Notebook interface shows three code cells. The first cell prints the 'persons' DataFrame. The second cell selects the 'city' column as a Series. The third cell selects two columns, 'age' and 'gender', as a DataFrame.

```
print (persons)
```

	name	age	gender	city
0	Maria	24	F	Patras
1	Christine	26	F	Athens
0	Vassilis	27	M	Arta
1	Betty	22	F	Sparta

```
In [3]: # select column 'city' (Series)
print (persons.city)
```

```
0 Patras
1 Athens
0 Arta
1 Sparta
Name: city, dtype: object
```

```
In [4]: # select two columns (DataFrame)
print (persons[['age', 'gender']])
```

	age	gender
0	24	F
1	26	F
0	27	M
1	22	F

In [ ]:

### Slicing

If we need to select a range of rows, we can specify the range using " : ". If we need to select a range of rows, using their labels we can use the method `loc`:

The Jupyter Notebook interface shows three code cells. The first cell creates a DataFrame and prints it. The second cell uses slicing to get rows by index. The third cell uses the 'loc' method to get rows by name and specific columns.

```
In [2]: import pandas as pd
columns = ['name', 'age', 'gender', 'city']
persons = pd.DataFrame([['Maria', 24, "F", "Patras"],
 ['Christine', 26, "F", "Athens"],
 ['Vassilis', 27, "M", "Arta"],
 ['Betty', 22, "F", "Sparta"]],
 index=[5,6,7,8],
 columns=columns)
print(persons)
```

	name	age	gender	city
5	Maria	24	F	Patras
6	Christine	26	F	Athens
7	Vassilis	27	M	Arta
8	Betty	22	F	Sparta

```
In [3]: # get rows with slicing using their index number
print(persons[1:3])
```

	name	age	gender	city
6	Christine	26	F	Athens
7	Vassilis	27	M	Arta

```
In [4]: # get rows with slicing using the row names
we also specify specific columns
print (persons.loc[0:6,['name','city']])
```

	name	city
5	Maria	Patras
6	Christine	Athens



In [ ]:

If we need to select a range of rows and/or columns, using their positions we can use method `iloc`:










```

df.iloc[0] # First row of a data frame
df.iloc[i] # (i+1)th row
df.iloc[-1] # Last row
df.iloc[:, 0] # First column
df.iloc[:, -1] # Last column
df.iloc[0:7] # First 7 rows
df.iloc[:, 0:2] # First 2 columns
df.iloc[1:3, 0:2] # 2nd through 3rd rows and first 2 columns
df.iloc[[0,5], [1,3]] # 1st and 6th rows and 2nd and 4th columns

```

 **jupyter** Slicing example (unsaved changes)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

         Code

```

In [6]: print(persons)

 name age gender city
5 Maria 24 F Patras
6 Christine 26 F Athens
7 Vassilis 27 M Arta
8 Betty 22 F Sparta

In [7]: persons.iloc[0] # object representing the first row
Out[7]: name Maria
 age 24
 gender F
 city Patras
 Name: 5, dtype: object



In [8]: persons.iloc[0].city # city value of first row: 'Patras'
Out[8]: 'Patras'

In [9]: persons.iloc[0,0] # first column value of first row: 'Maria'
Out[9]: 'Maria'










```

## Filtering

To subset the data we can apply Boolean indexing. This indexing is commonly known as a filter.

 **jupyter** Slicing example (unsaved changes)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

         Code

```

In [11]: print (persons [persons.age < 25]) # persons with age < 25

 name age gender city
5 Maria 24 F Patras
8 Betty 22 F Sparta

In [12]: # alternatively create a boolean Series first
youngBooleans = persons.age<25
print (youngBooleans)

5 True
6 False
7 False
8 True
 Name: age, dtype: bool

In [13]: # and use it to select the corresponding rows
young = persons[youngBooleans]
print(young)

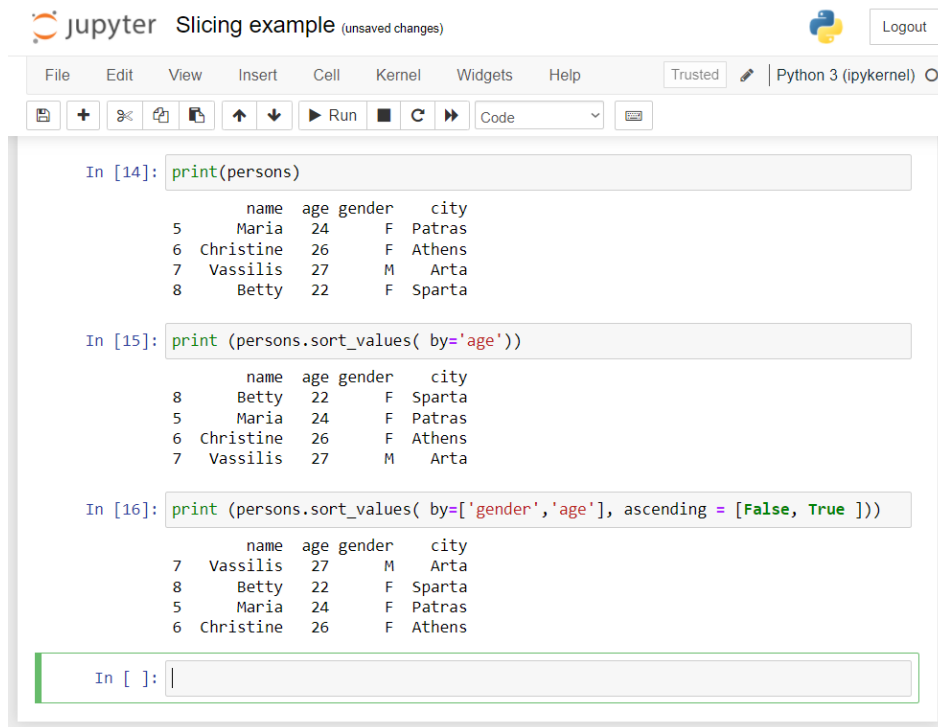
 name age gender city
5 Maria 24 F Patras
8 Betty 22 F Sparta

In []:

```

## 5. Sorting

We can use the `sort_values` method, to sort the data **by** a value in a column (or more than one columns). By default the sorting will occur in **ascending** order. `sort_values` will return a new DataFrame.



The image shows a Jupyter Notebook interface with the title "Slicing example (unsaved changes)". The notebook contains three code cells demonstrating the `sort_values` method on a DataFrame named `persons`.

**Cell 14:** `print(persons)`

	name	age	gender	city
5	Maria	24	F	Patras
6	Christine	26	F	Athens
7	Vassilis	27	M	Arta
8	Betty	22	F	Sparta

**Cell 15:** `print (persons.sort_values( by='age'))`

	name	age	gender	city
8	Betty	22	F	Sparta
5	Maria	24	F	Patras
6	Christine	26	F	Athens
7	Vassilis	27	M	Arta

**Cell 16:** `print (persons.sort_values( by=['gender','age'], ascending = [False, True ]))`

	name	age	gender	city
7	Vassilis	27	M	Arta
8	Betty	22	F	Sparta
5	Maria	24	F	Patras
6	Christine	26	F	Athens

The interface also shows a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar indicating "Trusted" and "Python 3 (ipykernel)".

## 6. Descriptive Statistics

Generating statistics is an easy task in Python as we have available numerous methods. Below you can see some of the most commonly used:

- **describe:** Statistics (count, mean, std, min, quantiles, max)
- **count:** Number of non-NA cells
- **sum:** Sum of the values
- **prod:** Value of the product
- **min, max:** Minimum and maximum values
- **mean, median, mode:** Arithmetic average, median and mode
- **var, std:** Variance and standard deviation
- **sem:** Standard error of mean
- **skew:** Sample skewness
- **kurt:** Kurtosis

The image shows a Jupyter Notebook interface with the title "Slicing example (unsaved changes)". The notebook contains two code cells. The first cell, labeled "In [18]:", executes the command `print(persons.describe())`, which outputs a summary of statistics for the 'age' column. The second cell, labeled "In [21]:", executes the command `print(persons.describe(include='all'))` with a comment `# Summarize all columns`, which outputs a more comprehensive summary including statistics for 'name', 'age', 'gender', and 'city'.

```
In [18]: print(persons.describe())
```

	age
count	4.000000
mean	24.750000
std	2.217356
min	22.000000
25%	23.500000
50%	25.000000
75%	26.250000
max	27.000000

```
In [21]: print(persons.describe(include='all'))
Summarize all columns
```

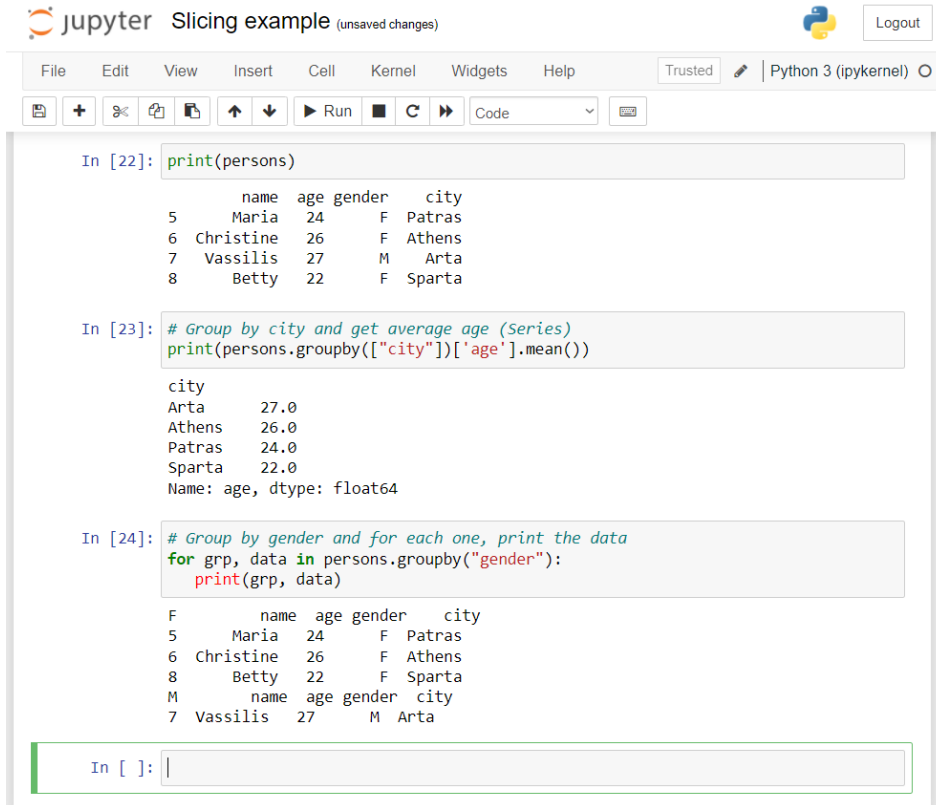
	name	age	gender	city
count	4	4.000000	4	4
unique	4	NaN	2	4
top	Maria	NaN	F	Patras
freq	1	NaN	3	1
mean	NaN	24.750000	NaN	NaN
std	NaN	2.217356	NaN	NaN
min	NaN	22.000000	NaN	NaN
25%	NaN	23.500000	NaN	NaN
50%	NaN	25.000000	NaN	NaN
75%	NaN	26.250000	NaN	NaN
max	NaN	27.000000	NaN	NaN

## 7. Group By

Using the **groupby** method we can:

- Split the data into groups based on some criteria
- Calculate statistics (or apply a function) to each group

Take a look at the following examples:



The image shows a Jupyter Notebook interface titled "Slicing example (unsaved changes)". The notebook contains three code cells demonstrating the use of the `groupby` method on a DataFrame named `persons`.

**Cell 1:** Prints the entire `persons` DataFrame.

```
In [22]: print(persons)
```

	name	age	gender	city
5	Maria	24	F	Patras
6	Christine	26	F	Athens
7	Vassilis	27	M	Arta
8	Betty	22	F	Sparta

**Cell 2:** Groups the data by city and calculates the average age for each city.

```
In [23]: # Group by city and get average age (Series)
print(persons.groupby(["city"])['age'].mean())
```

city	age
Arta	27.0
Athens	26.0
Patras	24.0
Sparta	22.0

Name: age, dtype: float64

**Cell 3:** Groups the data by gender and prints the data for each group.

```
In [24]: # Group by gender and for each one, print the data
for grp, data in persons.groupby("gender"):
 print(grp, data)
```

F	name	age	gender	city
5	Maria	24	F	Patras
6	Christine	26	F	Athens
8	Betty	22	F	Sparta

M	name	age	gender	city
7	Vassilis	27	M	Arta

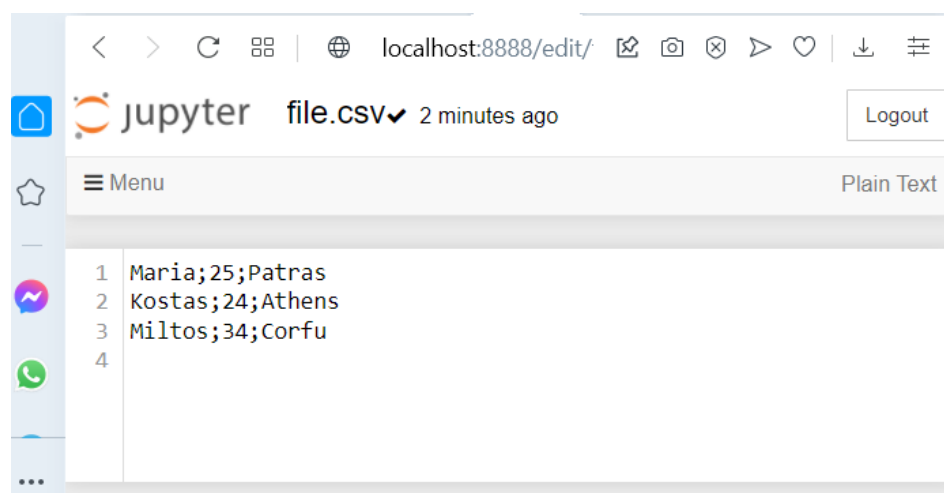
The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a status bar indicating the current kernel is Python 3 (ipykernel).

## 8. File I/O

There are many functions for reading/writing Pandas structures from/to different types of files and other sources. Here is a table of commonly used functions:

Format Type	Data Description	Reader	Writer
text	CSV	<b>read_csv</b>	<b>to_csv</b>
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
binary	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google Big Query	read_gbq	to_gbq

We give an example of reading a DataFrame from a CSV file using the functions in the first row of the table (in bold). We have created the csv file in jupyter using the new file (text file) option. There we insert our data and save the file as "file.csv":



Then, we return to the Notebook where we execute Python and try the code below:



**jupyter Slicing example** (unsaved changes) Python 3 (ipykernel) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [9]: `import pandas as pd`

In [10]: `t = pd.read_csv('file.csv',  
sep=';',  
names=[ 'name', 'age', 'city'],  
usecols=[ 'name', 'city'])`

In [11]: `print(t)`

	name	city
0	Maria	Patras
1	Kostas	Athens
2	Miltos	Corfu

In [ ]: |

Now, let's the reverse process: write a DataFrame as a CSV file.

**jupyter writeToCSV** (unsaved changes) Python 3 (ipykernel) Logout

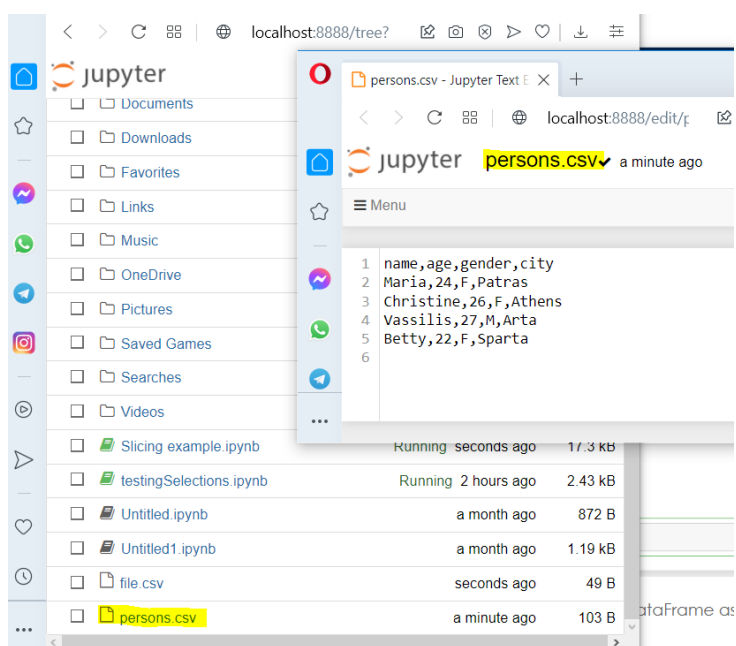
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

In [6]: `print(persons)`

	name	age	gender	city
0	Maria	24	F	Patras
1	Christine	26	F	Athens
0	Vassilis	27	M	Arta
1	Betty	22	F	Sparta

In [7]: `persons.to_csv('persons.csv', index=False)`

As you can see below, after we execute the `persons.to_csv` method a new file is created and we can see it in the Files menu of jupyter.



## 9. Missing values and duplicates

Missing values are marked as **NaN (Not a Number)**. There are several methods to deal with missing values in a DataFrame:

- **dropna()**: Drop missing observations
- **dropna(how='all')**: Drop observations where all cells are NA
- **dropna(axis=1, how='all')**: Drop column if all the values are missing
- **dropna(thresh = 5)**: Drop rows that contain less than 5 non-missing values
- **fillna(0)**: Replace missing values with zeros
- **isnull()**: Returns True if the value is missing
- **notnull()**: Returns True for non-missing values

Bellow you can see examples of using some of these methods.

The image shows a Jupyter Notebook interface with the following content:

**File Edit View Insert Cell Kernel Widgets Help**

Not Trusted Python 3 (ipykernel) Logout

**In [17]:**

```
import pandas as pd
t=pd.read_csv('mydata.csv', sep=';',
 names=['name', 'age', 'gender', 'city', 'height'])
```

**In [18]:**

```
print(t)
```

	name	age	gender	city	height
0	name	age	gender	city	height
1	Petros	19.5	M	Patras	171.0
2	Nikos	34	M	Athens	179.0
3	Betty	NaN	NaN	Corfu	172.0
4	Christina	45	F	Ioannina	NaN
5	Tomas	36	M	Patras	192.0

**In [19]:**

```
find missing values in a DataFrame
t.isnull() # DataFrame of Booleans
```

**Out[19]:**


	name	age	gender	city	height
0	False	False	False	False	False
1	False	False	False	False	False
2	False	False	False	False	False
3	False	True	True	False	False
4	False	False	False	False	True
5	False	False	False	False	False

**In [20]:**

```
count NaN values in each column (Series)
t.isnull().sum()
```












**Out[20]:**

```
name 0
age 1
gender 1
city 0
height 1
dtype: int64
```

jupyter writeToCSV  Logout

Not Trusted | Python 3 (ipykernel)

File Edit View Insert Cell Kernel Widgets Help

          Code 

```
city 0
height 1
dtype: int64
```

In [21]: *# find missing values in a Series*  
t.age.isnull()  
*# True if NaN, False otherwise (Series)*

Out[21]:

```
0 False
1 False
2 False
3 True
4 False
5 False
Name: age, dtype: bool
```

In [23]: t.dropna() *# drop a row if ANY values are missing*

Out[23]:

	name	age	gender	city	height
0	name	age	gender	city	height
1	Petros	19.5	M	Patras	171.0
2	Nikos	34	M	Athens	179.0
5	Tomas	36	M	Patras	192.0

## 10. Other data cleaning tasks

In the previous section, we discussed how to work with missing values and duplicates. Here we talk about other cleaning tasks we may execute before analyzing our data.

This 'cleaning' is often needed when our datasets have either column names that are not easy to understand, or unimportant information in the first few and/or last rows, such as definitions of the terms in the dataset, or footnotes. So, we need to rename columns and skip certain rows. Let's go through an example and work with the `olympics.csv`, a dataset with information on Olympic games, countries, number of gold, silver and bronze metals won and type of games (summer or winter).

First, we will read the csv file into a Pandas DataFrame:

Jupyter Notebook interface showing the first step of data loading. The code cell imports pandas and reads 'olympics.csv'. The output shows a DataFrame with 16 columns and 5 rows of data. The first row contains NaN and various symbols, which are not the intended headers.

```
In [4]: import pandas as pd

olympics_df = pd.read_csv('olympics.csv')
olympics_df.head()
```

Out[4]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	NaN	? Summer	01 !	02 !	03 !	Total	? Winter	01 !	02 !	03 !	Total	? Games	01 !	02 !	03 !	Combined total
1	Afghanistan (AFG)	13	0	0	2	2	0	0	0	0	0	13	0	0	2	2
2	Algeria (ALG)	12	5	2	8	15	3	0	0	0	0	15	5	2	8	15
3	Argentina (ARG)	23	18	24	28	70	18	0	0	0	0	41	18	24	28	70
4	Armenia (ARM)	5	1	2	9	12	6	0	0	0	0	11	1	2	9	12

If we take a closer look we see that columns are the string form of integers indexed at 0. The row which should have been our header (i.e. the one to be used to set the column names) is at `olympics_df.iloc[0]`. This happened because our CSV file starts with 0, 1, 2, ..., 15. Also, the NaN above should really be something like "Country", "? Summer" is supposed to represent "Summer Games", "01 !" should be "Gold", and so on. Therefore, we need to do two things:

- Skip one row and set the header as the first (0-indexed) row
- Rename the columns

We can skip rows and set the header while reading the CSV file by passing some parameters to the `read_csv()` function to remove the 0th row:

Jupyter Notebook interface showing the second step of data loading. The code cell reads 'olympics.csv' with `header=1`, skipping the first row. The output shows a DataFrame with 16 columns and 5 rows of data. The first row now contains the intended headers.


```
In [5]: olympics_df = pd.read_csv('olympics.csv', header=1)
olympics_df.head()
```

Out[5]:

	Unnamed: 0	? Summer	01 !	02 !	03 !	Total	? Winter	01 !.1	02 !.1	03 !.1	Total.1	? Games	01 !.2	02 !.2	03 !.2	Combined total
0	Afghanistan (AFG)	13	0	0	2	2	0	0	0	0	0	13	0	0	2	2
1	Algeria (ALG)	12	5	2	8	15	3	0	0	0	0	15	5	2	8	15
2	Argentina (ARG)	23	18	24	28	70	18	0	0	0	0	41	18	24	28	70
3	Armenia (ARM)	5	1	2	9	12	6	0	0	0	0	11	1	2	9	12
4	Australasia (ANZ) [ANZ]	2	3	4	5	12	0	0	0	0	0	2	3	4	5	12

To rename the columns, we use the DataFrame's `rename()` method, which allows us to relabel an axis based on a mapping (in this case, a dictionary).

Let's start by defining a dictionary that maps current column names (as keys) to more usable ones (the dictionary's values):

jupyter Untitled3 Last Checkpoint: 34 minutes ago (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Code

```
In [6]: new_names = {'Unnamed: 0': 'Country',
 '? Summer': 'Summer Olympics',
 '01 !': 'Gold',
 '02 !': 'Silver',
 '03 !': 'Bronze',
 '? Winter': 'Winter Olympics',
 '01 !.1': 'Gold.1',
 '02 !.1': 'Silver.1',
 '03 !.1': 'Bronze.1',
 '? Games': '# Games',
 '01 !.2': 'Gold.2',
 '02 !.2': 'Silver.2',
 '03 !.2': 'Bronze.2'}
```

```
In [8]: # We call the rename() function on our object:
olympics_df.rename(columns=new_names, inplace=True)
#Setting inplace to True specifies that our changes be made directly to the object
```

```
In [9]: # Let's see if everything worked out
olympics_df.head()
```

Out[9]:

	Country	Summer Olympics	Gold	Silver	Bronze	Total	Winter Olympics	Gold.1	Silver.1	Bronze.1	Total.1	# Games	Gold.2	Silver.2	Bronze.2	Combined total
0	Afghanistan (AFG)	13	0	0	2	2	0	0	0	0	0	13	0	0	2	2
1	Algeria (ALG)	12	5	2	8	15	3	0	0	0	0	15	5	2	8	15
2	Argentina (ARG)	23	18	24	28	70	18	0	0	0	0	41	18	24	28	70
3	Armenia (ARM)	5	1	2	9	12	6	0	0	0	0	11	1	2	9	12
4	Australasia (ANZ) [ANZ]	2	3	4	5	12	0	0	0	0	0	2	3	4	5	12

In [ ]:

## 11. Importing and manipulating large datasets

When working with big data, we need to import and handle files that may be too large (gigabytes or terabytes). It is a very handy thing that Pandas provides options for this purpose. Essentially, we are not importing the whole file in one go but partial contents.

### **dask.dataframe()**

The `dask.dataframe` is a collection of smaller pandas data frames split by the index (the row labels used for identification of data), which can be processed in parallel on a single machine or on multiple machines on a cluster. You can use it with the following Python script (put the name of your file in parentheses):

```
import dask.dataframe as dd
data = dd.read_csv("datafile.csv")
```

With `dask` the data is not fully loaded into memory but is ready to be processed. Also, certain operations can be performed again without loading the whole dataset into memory. Another advantage is that the most functions used with `pandas` can be also used with `dask`. The differences arise from the parallel nature of `dask`.


### **datatable**

`datatable` is a Python package for manipulating big 2-dimensional tabular data structures (meaning data frames, up to 100GB). To use the `datatable` package, on the other hand, you can try the following code:

```
import datatable as dt
data = dt.fread(input_file)
```

### **pandas.read\_csv()**

Another option offered by `Pandas` when dealing with big data, is the method `pandas.read_csv()` which can be run with the `chunksize` option. This will break the input file into chunks instead of loading the whole file into memory. This will reduce the pressure on memory for large input files and given an optimal `chunksize` found through trial and error, there can be significant increase in efficiency. The code below will split the input file into chunks of 100 000 lines.

jupyter workingBigData Last Checkpoint: an hour ago (autosaved)  Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
In [13]: import pandas as pd
import numpy as np
import pandas.util.testing
setting the number of rows for the CSV file we will create
N = 1000000

creating a pandas dataframe (df) with 8 columns and N rows
with random integers between 999 and 999999 and with column names from A to H
df = pd.DataFrame(np.random.randint(999,999999,size=(N, 7)), columns=list("ABCDEFGH"))

creating one column 'H' of float type using the uniform distribution
df["H"] = np.random.rand(N)

creating two additional columns with random strings
df["I"] = pd.util.testing.rands_array(10, N)
df["J"] = pd.util.testing.rands_array(10, N)

print the dataframe to see what we have created
print(df)
export the dataframe to csv using comma delimiting
df.to_csv("random.csv", sep=",")
```

	A	B	C	D	E	F	G	H	\
0	839205	846420	409821	807551	357070	81729	360556	0.976302	
1	613013	283569	373392	301300	936988	705113	433219	0.292355	
2	674611	471695	71170	196722	575455	373687	745315	0.789196	
3	930781	597155	421093	71397	242798	904452	114565	0.672768	
4	289620	409845	192041	298712	304673	431307	343416	0.851714	
...	...	...	...	...	...	...	...	...	...
999995	1038	997477	26388	248213	441945	536098	622296	0.072140	
999996	252941	285337	700570	250081	144600	675368	232869	0.862313	
999997	358727	815912	338754	240715	264473	466331	645510	0.560346	
999998	118626	141699	677775	428739	608437	815107	251088	0.577492	
999999	167869	582895	791287	830604	645231	550140	668590	0.428766	
...									
	I		J						
0	aYuBMJJv3W		Kz7MomtFZX						
1	hHzO8GsMki		kA9Flily3J						
2	PrzhMORNs9		h849ldiebB						
3	am8TGyDbir		60E254kDFs						
4	LmftdESEaa		JjcqX5rRxh						
...	...		...						
999995	YcFuFVLhal		hQZRjtclQ6						
999996	5ipkiyGQkh		44ort3IwpM						
999997	riBWLl712y		eJC7PhgNkv						
999998	biuiRSbGfM		NN3LXf4kNv						
999999	BLnsO9sTOj		9IFcdWQYjJ						
999997	358727	815912	338754	240715	264473	466331	645510	0.560346	
999998	118626	141699	677775	428739	608437	815107	251088	0.577492	
999999	167869	582895	791287	830604	645231	550140	668590	0.428766	
...									
	I		J						
0	aYuBMJJv3W		Kz7MomtFZX						
1	hHzO8GsMki		kA9Flily3J						
2	PrzhMORNs9		h849ldiebB						
3	am8TGyDbir		60E254kDFs						
4	LmftdESEaa		JjcqX5rRxh						
...	...		...						
999995	YcFuFVLhal		hQZRjtclQ6						
999996	5ipkiyGQkh		44ort3IwpM						
999997	riBWLl712y		eJC7PhgNkv						
999998	biuiRSbGfM		NN3LXf4kNv						
999999	BLnsO9sTOj		9IFcdWQYjJ						

[1000000 rows x 10 columns]

```
In [14]: # The code below will split the input file into chunks of 100 000 lines
chunks = pd.read_csv("random.csv", chunksize=100000)

After reading the data chunk by chunk, we must concatenate them
data = pd.concat(chunks)
```

# The Matplotlib library in Python for data visualization

Site: [DTAM Online Training Platform](#)

Course: Big Data

Book: The Matplotlib library in Python for data visualization

Printed by: Vasileios Gkamas

Date: Friday, 27 October 2023, 4:20 PM



## Table of contents

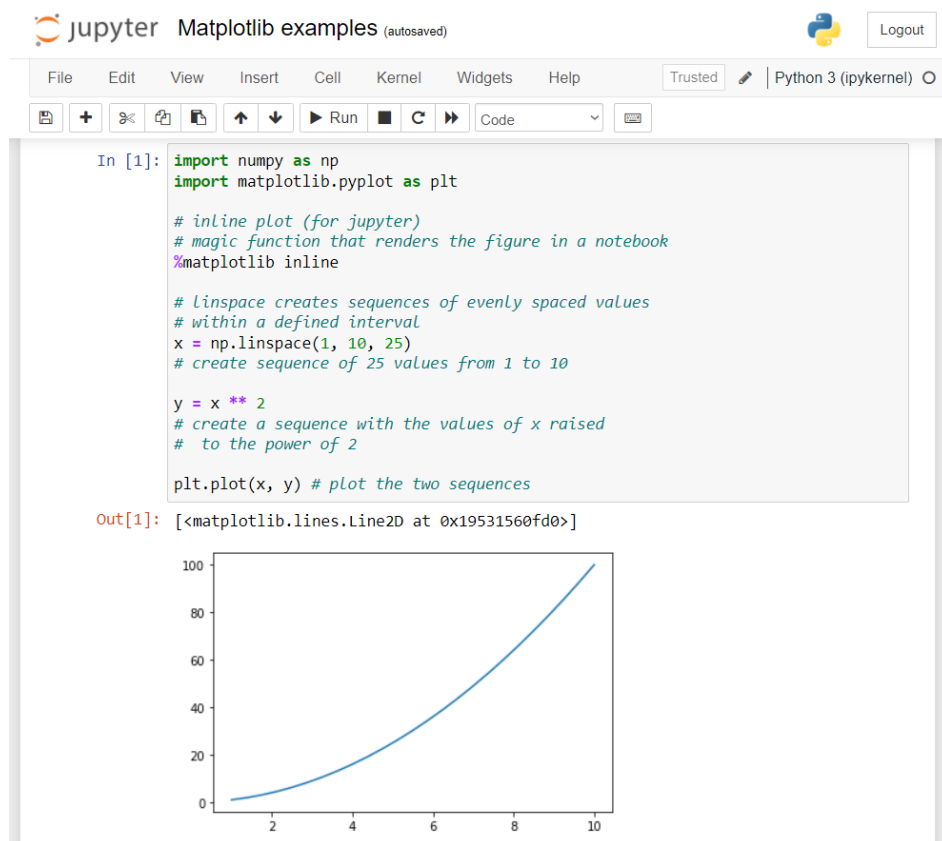
1. Creating Basic Plots
2. Creating scatter 2D plots
3. Histograms and Density Plots

# 1. Creating Basic Plots

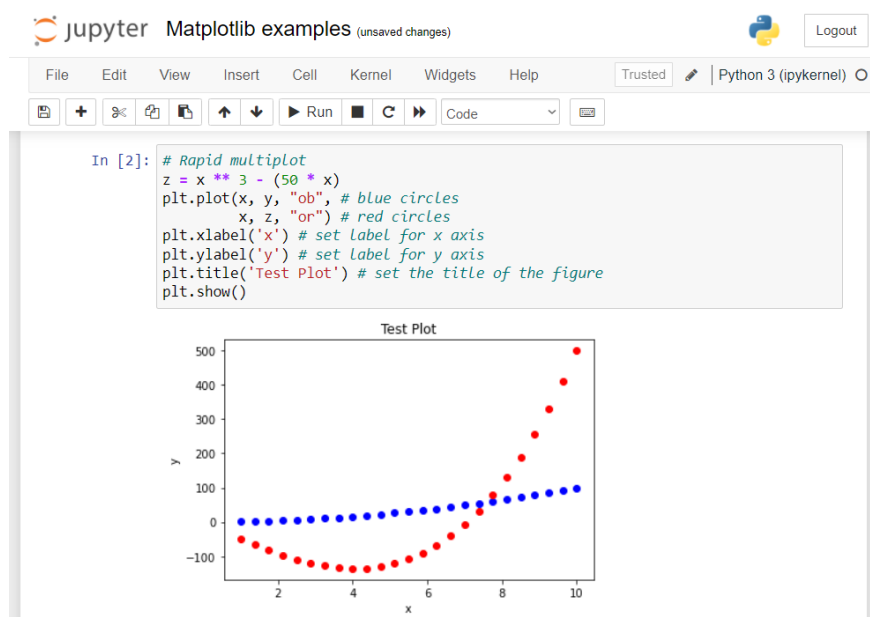
*Matplotlib* is a Python 2D plotting library written for Python. It can produce high-quality graphs in a variety of formats. It consists of **pyplot**, an object-oriented interface to the plotting library.

In a matplotlib plot, the **figure** is the canvas on which all drawing components are plotted. The figure consists of **axes**, that are subdivisions of the figure. Each of the axes consists of one or more **axis** (horizontal x, vertical y or even depth z).

The **plot function** takes as arguments one or more pairs of sequences for the x and y axis. For every x, y pair of arguments, there is an optional third argument which is the format string that indicates the color and line type of the plot. The default format string is 'b-', which is a solid blue line.



In the following example we provide three additional parameters to the plot function for an additional plot line.

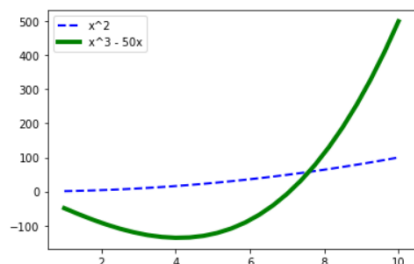


Alternatively, we can use multiple calls of the plot function. We can specify parameters such as: **label**, **color**, **linestyle**, **linewidth**

```
In [3]: plt.plot(x, y, label='x^2', color='blue',
 linestyle='--', linewidth=2)

plt.plot(x, z, label='x^3 - 50x', color='green',
 linestyle='-', linewidth=4)

plt.legend()
plt.show()
```



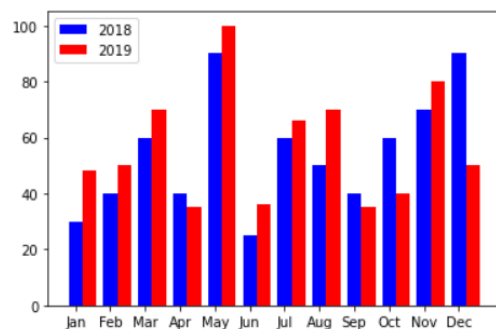
We can use the **bar** function instead of **plot**, to generate bar charts. If we want multiple bars, we can offset the **x** axis by adding some value. We can use **xticks** and **yticks** to specify specific labels for the points in the x and y axis.

```
In [4]: import numpy as np
import matplotlib.pyplot as plt
x = np.arange(12) # [0 1 2 3 4 5 6 7 8 9 10 11]

Performance data for each month of two years that we
want to compare
y = [30, 40, 60, 40, 90, 25, 60, 50, 40, 60, 70, 90]
z = [48, 50, 70, 35, 100, 36, 66, 70, 35, 40, 80, 50]

Plot the bars with specific width. Offset the second
by that width
width = 0.4
plt.bar(x, y, label='2018', color='blue', width=width)
plt.bar(x + width, z, label='2019', color='red', width=width)

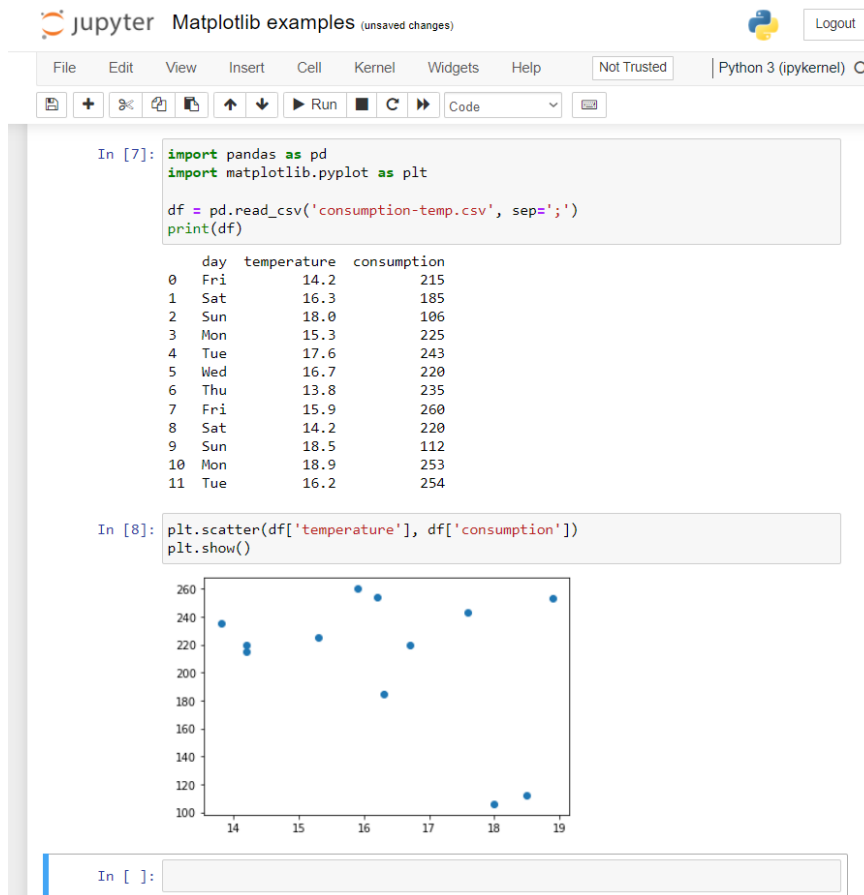
Specify labels for x-axis
x_labels = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep',
 'Oct', 'Nov', 'Dec']
plt.xticks(ticks=x, labels=x_labels)
plt.legend()
plt.show()
```



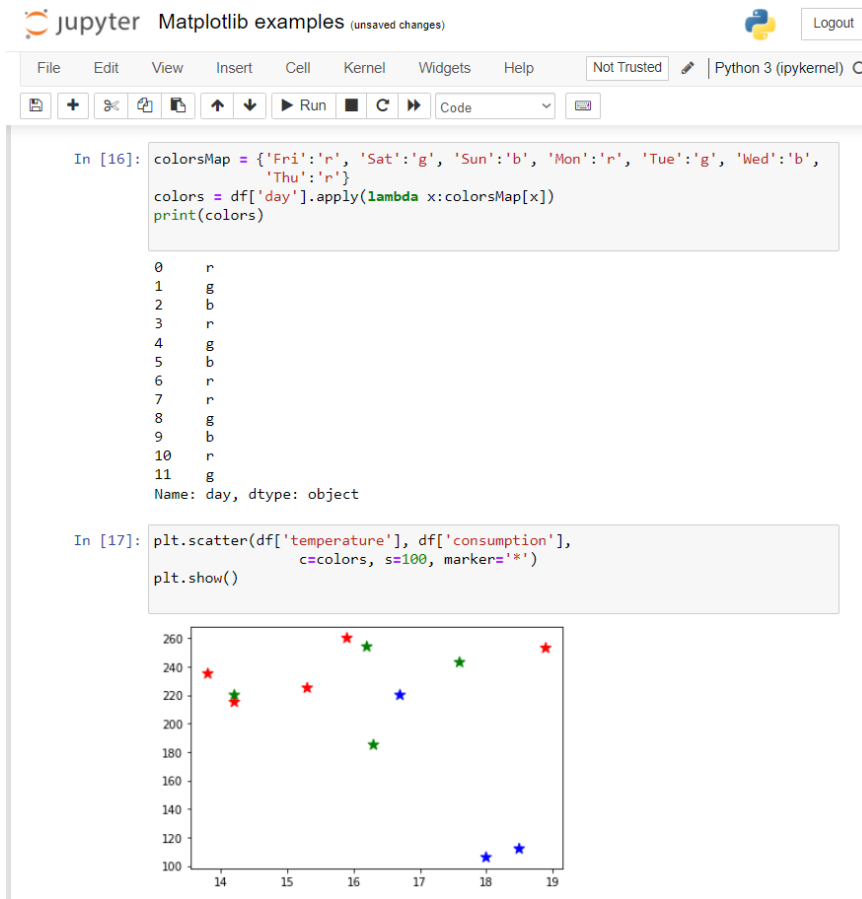
## 2. Creating scatter 2D plots

We can use the **scatter** function, to generate scatter plots. We can specify parameters like: label, color, marker, s (size).

In the following example we read a DataFrame from a csv file and create a scatter plot:



An additional example where we use different colors based on one of the columns of the dataframe and set the symbol to star:

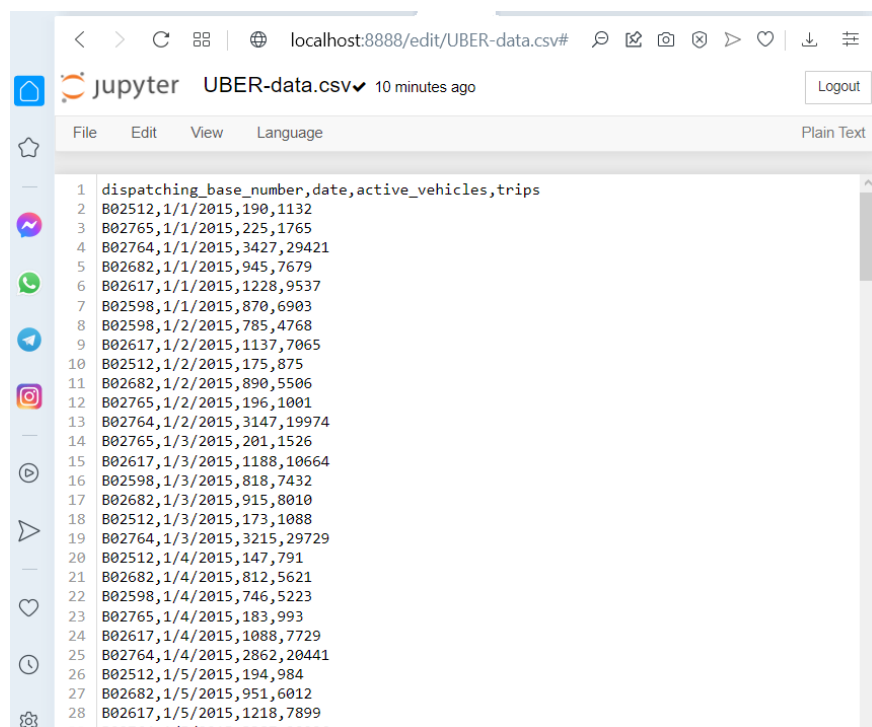


### 3. Histograms and Density Plots

**Histograms** and **Density Plots** can be a great first step in understanding a dataset. A histogram is an accurate graphical representation of the distribution of numerical data. It is an estimate of the probability distribution of a continuous variable. A Density Plot is a smoothed version of the histogram and is used in the same concept.

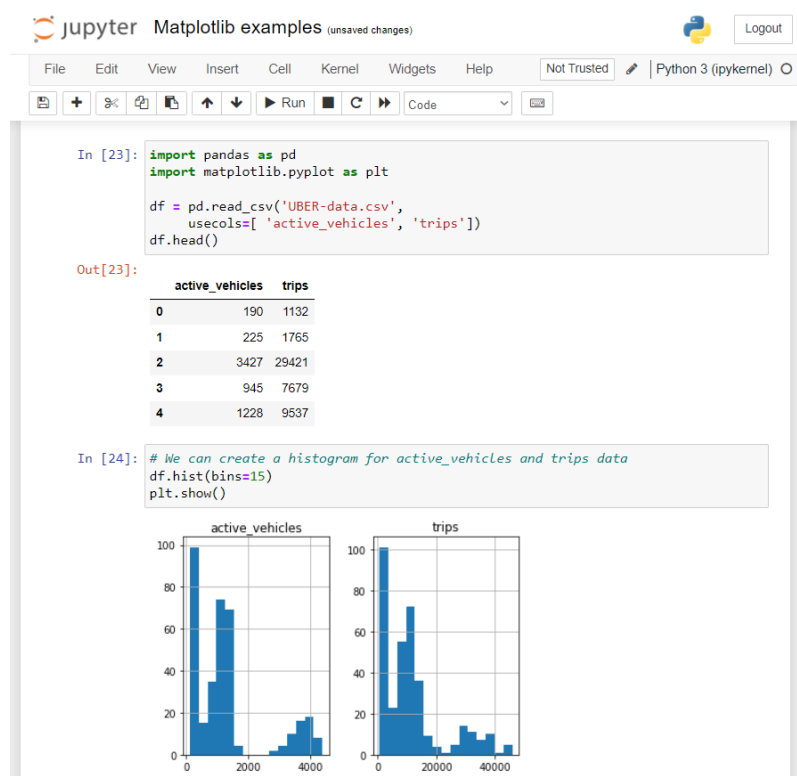
The **Pandas** library offers some plotting methods for DataFrames. The **hist** method can be used to generate histograms of the columns. The **plot** method has parameters we can use to generate density plots.

Let's assume we have a dataset with UBER data as depicted in the following screenshot:

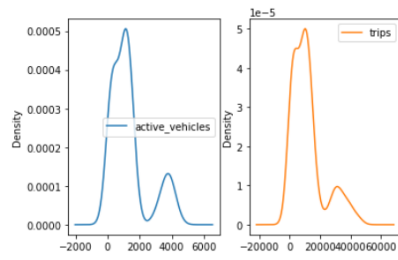


dispatching_base_number	date	active_vehicles	trips
B02512	1/1/2015	190	1132
B02765	1/1/2015	225	1765
B02764	1/1/2015	3427	29421
B02682	1/1/2015	945	7679
B02617	1/1/2015	1228	9537
B02598	1/1/2015	870	6903
B02598	1/2/2015	785	4768
B02617	1/2/2015	1137	7065
B02512	1/2/2015	175	875
B02682	1/2/2015	890	5506
B02765	1/2/2015	196	1001
B02764	1/2/2015	3147	19974
B02765	1/3/2015	201	1526
B02617	1/3/2015	1188	10664
B02598	1/3/2015	818	7432
B02682	1/3/2015	915	8010
B02512	1/3/2015	173	1088
B02764	1/3/2015	3215	29729
B02512	1/4/2015	147	791
B02682	1/4/2015	812	5621
B02598	1/4/2015	746	5223
B02765	1/4/2015	183	993
B02617	1/4/2015	1088	7729
B02764	1/4/2015	2862	20441
B02512	1/5/2015	194	984
B02682	1/5/2015	951	6012
B02617	1/5/2015	1218	7899

In the following example we first read two columns from this csv file, as a Dataframe. Let's assume we are interested in columns `active_vehicles` and `trips`.



```
In [25]: # ...or we can create a density plot (one for each selected column)
df.plot(kind='density', subplots=True,
 layout=(1,2), sharex=False)
plt.show()
```



```
In []: |
```

# Basic concepts of Apache Hadoop

Site: [DTAM Online Training Platform](#)  
Course: Big Data  
Book: Basic concepts of Apache Hadoop

Printed by: Vasileios Gkamas  
Date: Friday, 27 October 2023, 4:20 PM



# Table of contents

## **1. Introduction to Hadoop**

## **2. Hadoop advantages and disadvantages**

## **3. The Hadoop Architecture and major components of Hadoop ecosystem**

## **4. Hadoop clusters**

### 4.1. Hadoop master-slave topology

### 4.2. Single-node vs multi-node Hadoop clusters

### 4.3. Building an efficient Hadoop cluster

# 1. Introduction to Hadoop

Apache Hadoop is an open-source distributed processing framework that manages data processing and storage for big data applications in scalable clusters of servers. Hadoop is designed to scale up from single servers to thousands of machines, each offering local computation and storage. The library itself is designed to detect and handle failures at the application layer rather than relying on hardware to deliver high availability.

Hadoop aims to address the limitations of conventional Relational Database Management Systems (RDBMS) in terms of storing large datasets, handling data of different formats and processing data at high speed. More specific, a) RDBMS are not efficient for storing huge amount of data, as the cost of data storage is high given that it is related both to the hardware and software cost, b) RDBMS can store and manipulate structured data, however they fail with unstructured data; however big data coming from the real world are in a structured, unstructured and semi-structured format and c) as the data for processing are at the order of terabytes or petabytes, there is need for high speed processing in real-time where the traditional RDBMS fail.

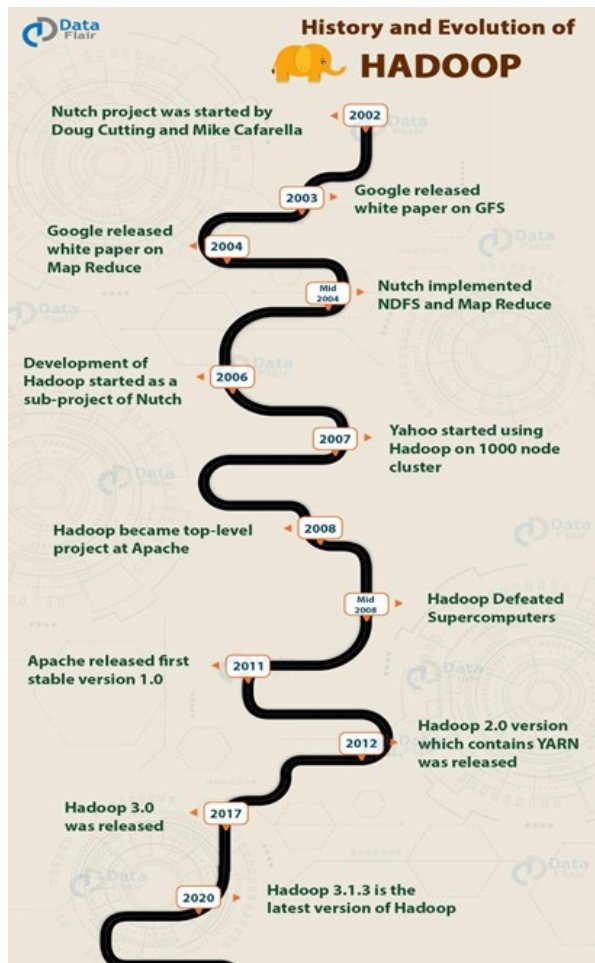


Figure 3.1: History and evolution of Hadoop (source: <https://data-flair.training/blogs/hadoop-history/>)

## 2. Hadoop advantages and disadvantages

Like any framework, Hadoop has both advantages and disadvantages:

### Hadoop Advantages

- **Quick processing of huge volumes of data** which is key considering the data that are generated from social media and the Internet of Things.
- **Varied Data Sources.** Hadoop supports a variety of data including structured and unstructured data (e.g. text and images). Data can come from a range of sources like email conversation, social media etc. Hadoop accepts data in various formats, such as text file, XML file, images, CSV files etc.
- **Fault tolerance.** Data and application processing are protected against hardware failure. If a node goes down, jobs are automatically redirected to other nodes to make sure the distributed computing does not fail. Multiple copies of all data are stored automatically.
- **Flexibility.** Unlike traditional relational databases, you don't have to preprocess data before storing it. You can store as much data as you want and decide how to use it later.
- **Low cost.** Given that Hadoop is an open-source framework it is free. Moreover, it runs on commodity hardware.
- **Scalability.** You can easily grow your Hadoop cluster to handle more data by adding new nodes. The more computing nodes you use, the more processing power you have. Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.
- **High throughput.** Hadoop stores data in a distributed manner allowing the easy use of distributed processing. A given job is divided into small jobs which work on chunks of data in parallel, thus giving high throughput.

### Hadoop disadvantages

- **Issue with small files.** Hadoop works efficiently when processing a small number of large files. But when it comes to the application which deals with a large number of small files, it is inefficient. A small file is nothing but a file which is significantly smaller than Hadoop's block size which can be either 128MB or 256MB by default.
- **Vulnerable by nature.** Hadoop is written in java. While java is a widely used programming language, it is easily exploited by making Hadoop vulnerable to cyberattacks.
- **Security.** To secure Hadoop, the Kerberos authentication is used whose management is complicated. Moreover, Hadoop does not support encryption at storage and network levels which are a major point of concern.
- **Processing overhead.** Hadoop cannot do in-memory calculations hence it incurs processing overhead. Instead in Hadoop the data is read from the disk and written to the disk which makes read/write operations very expensive when we are dealing with data size of terabytes and petabytes.
- **Supports only batch processing.** At the core, Hadoop has a batch processing engine which is not efficient in stream processing. It cannot work with real-time data streams with low latency. We have to collect in advance and store the data in a file which will be processed later by Hadoop.

### 3. The Hadoop Architecture and major components of Hadoop ecosystem

In fact, Hadoop is a collection of multiple tools and frameworks for big data management, storage, processing and analysis. There are 3 major components of Hadoop ecosystem:

- **Hadoop Distributed File System (HDFS)**. This is the storage unit of Hadoop. HDFS splits the data unit into smaller units that are called blocks and stores them in a distributed manner.
- **Hadoop MapReduce**. This is the processing unit of Hadoop allowing you to write applications for processing big data. MapReduce runs these applications in parallel on a cluster of low-end machines. It does so in a reliable and fault-tolerant manner.
- **Hadoop Yet Another Resource Negotiator (YARN)**. This is the resource management unit of Hadoop. It allocates to applications RAM, memory, and other resources depending on their requirements. The basic principle behind YARN is to separate resource management and job scheduling/monitoring functions into separate daemons.

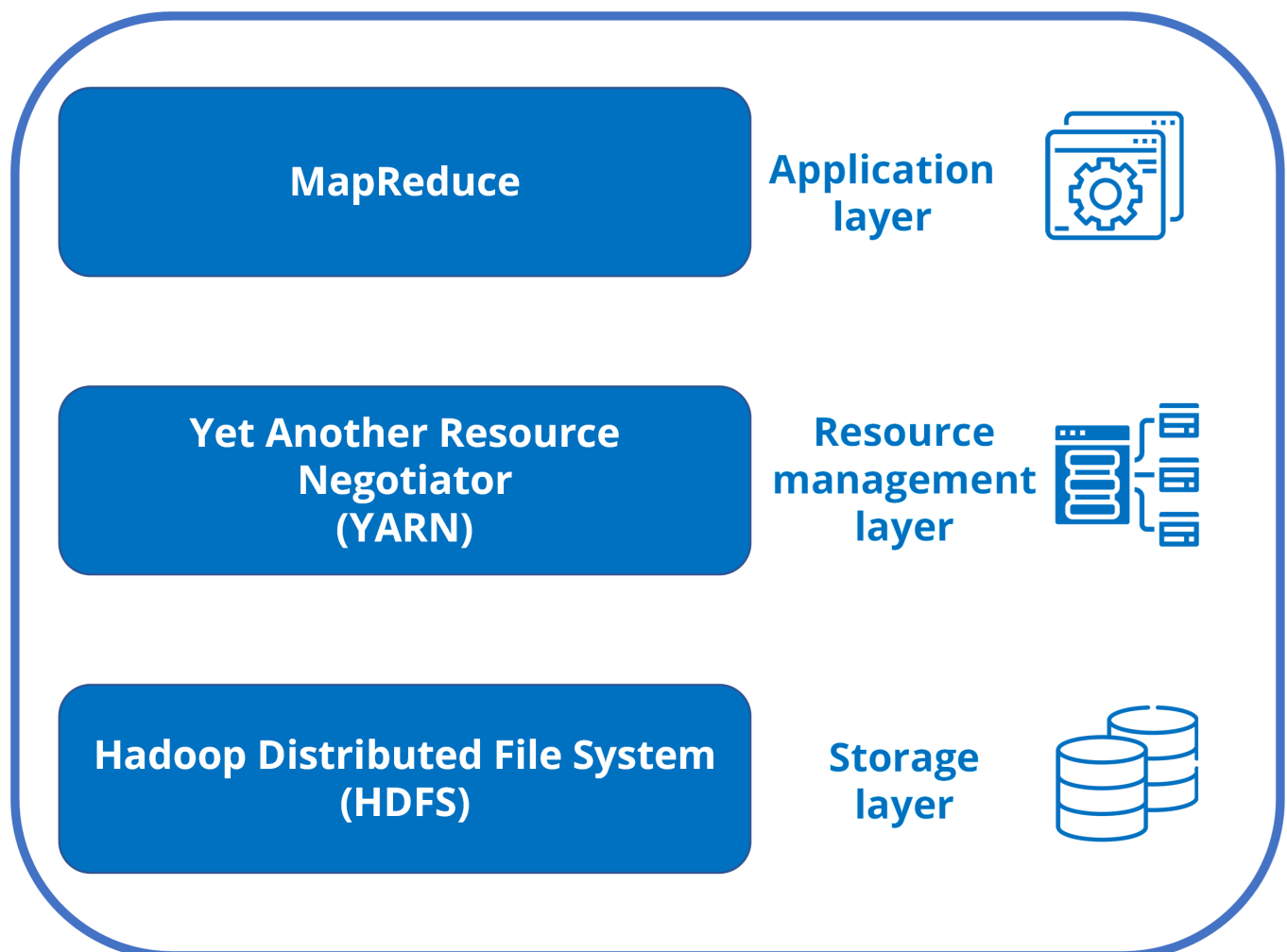


Figure 3.2: Hadoop architecture

Although HDFS, YARN and MapReduce consist of the main components of the Hadoop ecosystem, there are many other tools that are used to supplement or support these major components. Each of these tools is developed to deliver a concrete function and all work together to provide services such as data absorption, data analysis, data storage, data maintenance of data, etc.

In addition, to HDFS, YARN and MapReduce the other components of Hadoop ecosystem are the following:

- **Apache Spark**. It is an open-source unified analytics engine for large-scale data processing. Spark provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. It has built-in libraries for streaming, SQL, machine learning (like MLlib) and graph processing.
- **Apache Pig**. It is a SQL-like language used for querying and analyzing data stored in HDFS. Pig works both with structured and unstructured data, it is easily extensible -users can create their own functions to do special-purpose processing- and it also permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency. Pig does the work of executing commands and in the background all the activities of MapReduce are taken care of. After the processing, the Pig stores the result in HDFS. Yahoo was the original creator of the Pig.

- **Apache Hive.** It is a data warehouse software for reading, writing, and managing large datasets residing in distributed storage using SQL. Similar to Query Processing frameworks, Hive has two components: Java Database Connectivity/Open Database Connectivity (JDBC/ODBC) Drivers and Hive Command Line. Hive Command line is an interface for execution of Hive Query Language commands, while JDBC/ODBC establish the connection with the data storage. Facebook designed Hive for people who are comfortable in SQL.
- **Apache HBase.** It is a NoSQL database built on the top of HDFS. It is an open source, distributed, versioned, non-relational database modeled after Google's Bigtable. At times where we need to search or retrieve the occurrences of something small in a huge database, the request must be processed within a short time.
- **Apache Mahout.** It is an open source tool for creating scalable machine learning applications. It provides various libraries or functionalities such as collaborative filtering, clustering, and classification which are essential for developing Machine learning applications.
- **Spark MLlib.** It is a machine learning library aiming to make practical machine learning scalable and easy. It consists of common learning algorithms and utilities, including classification, regression, clustering, collaborative filtering, dimensionality reduction, as well as lower-level optimization primitives and higher-level pipeline APIs.
- **Apache Solr** and **Apache Lucene.** These are the two services that perform searching and indexing in Hadoop with the help of some java libraries. Apache Lucene is based on java, while Apache Solr is an application built around Apache Lucene.
- **Apache Zookeeper.** A big issue with Hadoop is the management, coordination and synchronization among its resources and components. Zookeeper overcomes these challenges by performing synchronization, inter-component based communication, grouping, and maintenance.
- **Apache Oozie.** It is a workflow scheduler system to manage Apache Hadoop jobs. Oozie Workflow jobs are Directed Acyclic Graphs (DAGs) of actions.

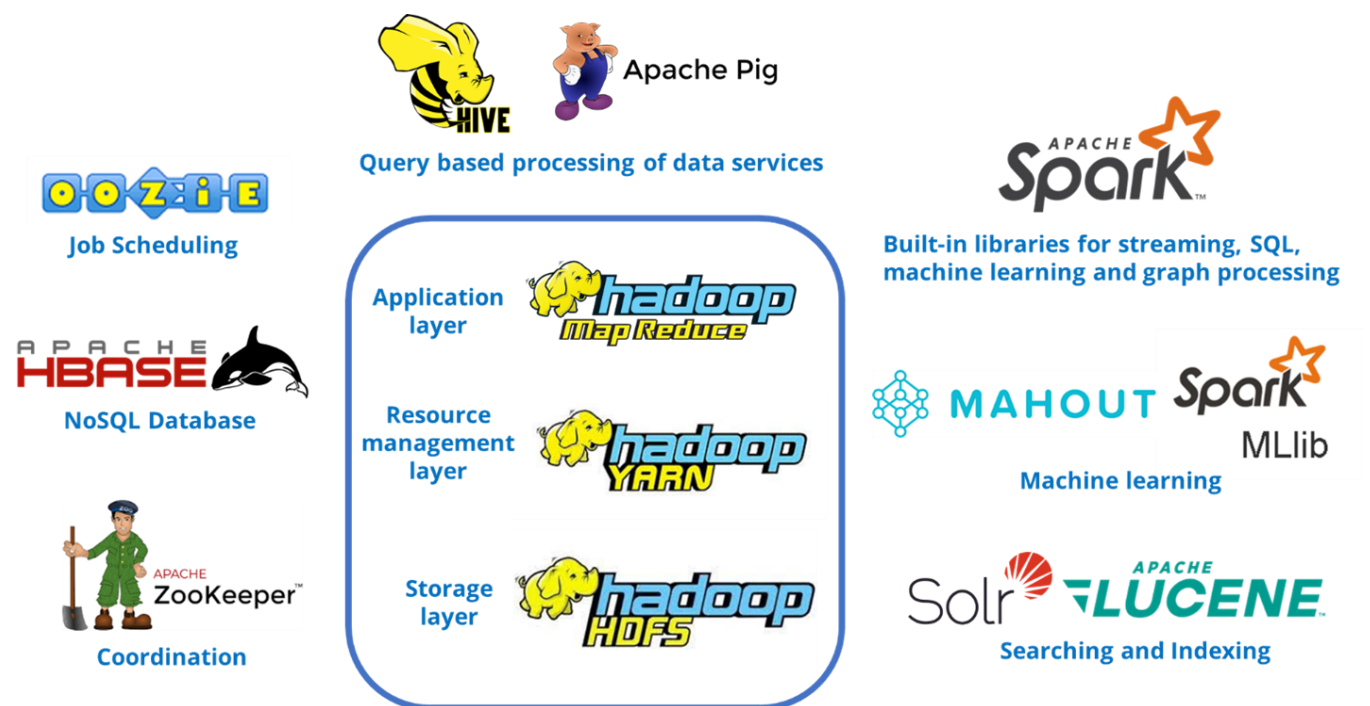


Figure 3.3: The Hadoop ecosystem

## 4. Hadoop clusters

A Hadoop cluster is a group of commodity hardware running the Hadoop framework for storing and analyzing huge amounts of data in a distributed computing environment. The machines of the Hadoop cluster are connected together via a local area network. The cluster follows a **master-slave topology**. The master node assigns a task to various slave nodes and manages resources, while slave nodes do the actual computing. Slave nodes store the real data whereas on master we have metadata.

Hadoop can run in three different modes:

- **Standalone (or local) mode.** There are no daemons running and everything runs as a single java process. The Standalone mode is suitable for running MapReduce programs during development, since it is easy to test and debug them.
- **Pseudo Distributed mode.** The Hadoop daemons run on the local machine, thus simulating a cluster on a small scale. At this mode, each Hadoop daemon runs in a separate Java process
- **Fully distributed mode.** The Hadoop daemons run on a cluster of machines.

## 4.1. Hadoop master-slave topology

### The master node

The master node is a high-power machine with strong specifications in terms of CPU and memory. Two daemons are running on the master node, the **NameNode** daemon and **Resource Manager** daemon.

NameNode is a master node in the Hadoop HDFS that manages the filesystem namespace. The NameNode daemon stores the metadata in the memory for fast retrieval. Its main functions are the following:

- Management of the file system namespace and access to files by the clients
- Store metadata of actual data. For example, such metadata are the file path, the number of blocks and the block id.
- Execution of file system namespace operations, such as opening, closing and renaming files/directories.

The **Resource Manager** is the master daemon of YARN. It arbitrates resources among competing nodes and keeps track of live and dead nodes.

### The slave nodes

A slave node in a Hadoop cluster is a low-cost commodity machine with a common configuration in terms of CPU and memory. A slave node runs two different daemons, the **DataNode** and the **Node Manager**. The DataNode daemon stores the data, executes read, write and data processing operations and creates, deletes and replicates data blocks once this is requested by the master node. The NodeManager daemon runs services on the node to check its health and reports the health of the slave node to the ResourceManager daemon running at the master node.

### The client nodes

Client Nodes in Hadoop are neither master node nor slave nodes. A Client node has Hadoop installed with all the required cluster configuration settings and is responsible for loading all the data into the Hadoop cluster. A Client node loads the data on the Hadoop cluster, tells how to process the data by submitting MapReduce jobs and collects the output from a specified location.

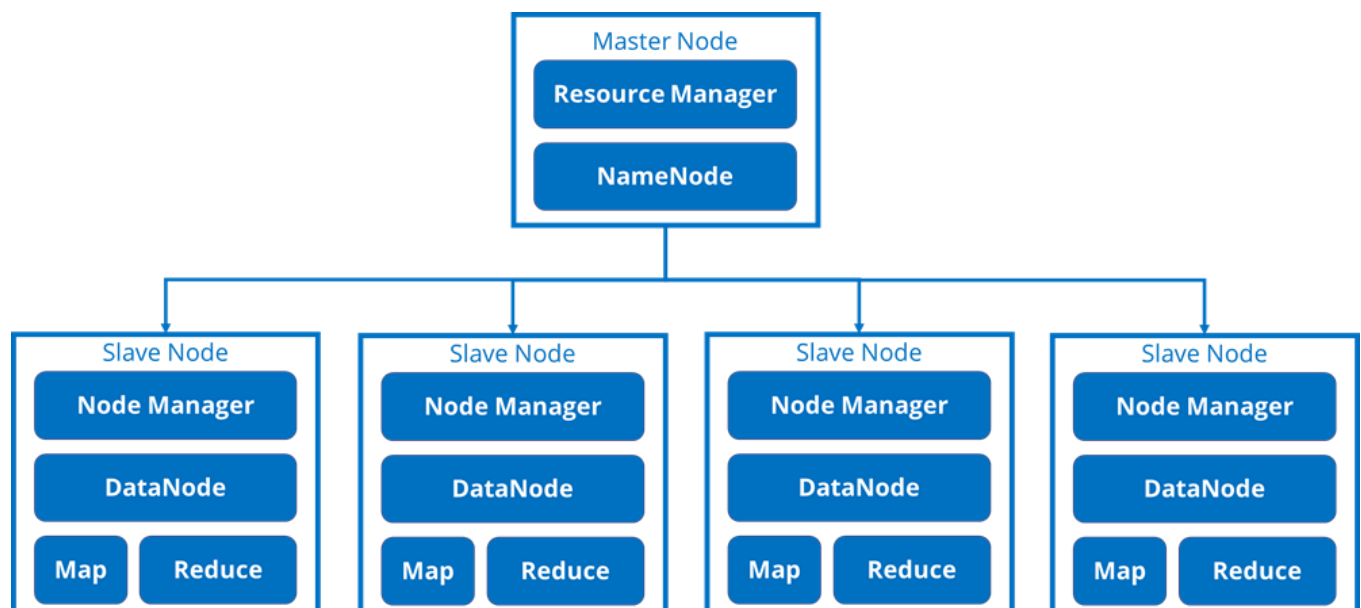


Figure 3.4: The master-slave topology in Hadoop cluster

## 4.2. Single-node vs multi-node Hadoop clusters

Hadoop can run both at **single-node clusters** and **multi-node clusters**.

A single-node cluster is deployed at a single machine where all the daemons, such as the DataNode and the NameNode are running. All the processes of a single node Hadoop cluster run on one JVM instance and the user needs only to set the JAVA\_HOME variable (there is no need for any additional configuration).

A multi-node cluster is deployed at several machines and the daemons are running on separate machines. This type of cluster follows the master-slave architecture where the NameNode daemon runs on the master machine and the DataNode daemon runs on the slave machines. In a multi-node cluster the daemons running at slave nodes, such as the DataNode and the Node Manager run on low-cost machines, while the daemons running at master nodes, such as the NameNode and the Resource Manager run on powerful and costly servers. In a multi-node Hadoop cluster, the slave nodes can be present in any location regardless the physical location of the master node.



## 4.3. Building an efficient Hadoop cluster

Some best practices in order to build an efficient Hadoop cluster are the following:

- **Choosing the right hardware for a Hadoop cluster** is a challenge, as many organizations have doubts about what type of hardware to use in order to deploy an effective Hadoop cluster. As no ideal cluster configuration, nor specific hardware specifications exist to setup a Hadoop cluster, the setup should provide a balance between economy and performance in order to serve a specific workload. However, some questions that should be addressed are the following: a) the volume of the data to be processed and stored, b) the type of workload that needs to be processed (CPU driven vs IO Bound), c) the data storage methodology (data container, data compression technique used , if any); and the data retention policy (How long can you afford to keep the data before deleting it).
- **Sizing a Hadoop cluster.** The volume of the data to be processed is a key parameter to identify how many nodes would be required to process the data efficiently, as well as how much memory will be required at each machine. The best practice is to size the cluster based on the amount of storage required.
- **Configuring the Hadoop Cluster.** Finding the ideal configuration for a Hadoop cluster is not easy, as Hadoop needs to be adapted to the cluster it is running and also to the job. The best way to decide on the ideal configuration for the cluster is to run the Hadoop jobs with the default configuration available to get a baseline. After that the job history log files can be analyzed to see if there is any resource weakness or if the time taken to run the jobs is higher than expected. Repeating the same process can help fine tune the Hadoop cluster configuration in such a way that it best fits the business requirements.

The main features of a good Hadoop cluster management tool are the following:

- It should provide diverse workload management, security, resource provisioning, performance optimization, health monitoring, policy management, job scheduling, backup and recovery across one or more nodes.
- It should implement NameNode high availability with load balancing and auto-failover.
- It should support policy-based management that prevents any application from consuming more resources than others.
- It should manage the deployment of any layers of software over Hadoop clusters by performing regression testing.

## Exercise: Hadoop installation on Ubuntu VM

Site: [DTAM Online Training Platform](#)  
Course: Big Data  
Book: Exercise: Hadoop installation on Ubuntu VM

Printed by: Vasileios Gkamas  
Date: Friday, 27 October 2023, 4:20 PM

## Table of contents

1. General description
2. Desired objectives
3. Required material
4. Other requirements
5. Solution

## 1. General description

This exercise concerns the installation of Hadoop at Linux environment. More specific we will create an Ubuntu 20.04.3 LTS Virtual Machine (VM) running Hadoop 3.3.1. For the creation of the VM we will use the VMWare workstation player. At the required material section of the exercise you can find the URLs for downloading the VMWare workstation player and the Ubuntu image.

## 2. Desired objectives

The desired objectives of the exercise are the following:

- Setting up a single node Hadoop Cluster in a pseudo-distributed mode
- Get familiar with the main configuration files of Hadoop
- Get familiar with the main administration tasks of Hadoop
- Get familiar with the management of Hadoop daemons

### 3. Required material

For the execution of this task the VMWare Workstation Player and the Ubuntu image are required. These are available at the following links:

- [URL](#) for downloading the VMware Workstation 16.2.0 Player:
- [URL](#) for downloading Ubuntu 20.04.3 LTS

This [URL](#) provides instructions for the creation of Ubuntu VM at VMware workstation:

Other useful material

- Apache Hadoop, [Hadoop: Setting up a Single Node Cluster](#).
- Tom White, Hadoop The Definitive Guide, STORAGE AND ANALYSIS AT INTERNET SCALE, 4<sup>th</sup> edition, O'Reilly, ISBN: 978-1-491-90163-2

## 4. Other requirements

There are no other special requirements for the execution of this exercise.

## 5. Solution

### Step 1: System update

Ensure that the Ubuntu system is updated

```
$sudo apt-get update
```

### Step 2: Java installation

Make sure that you have installed java. To check if java is already installed run the following command.

```
$ java -version
```

```
openjdk version "1.8.0_292"
```

```
OpenJDK Runtime Environment (build 1.8.0_292-8u292-b10-0ubuntu1~16.04.1-b10)
```

```
OpenJDK Server VM (build 25.292-b10, mixed mode)
```

If java is not installed, run the following commands for the installation of java and OpenJDK

```
$ sudo apt install default-jre
```

```
$ sudo apt install openjdk-11-jre-headless
```

```
$ sudo apt install openjdk-8-jre-headless
```

```
$ sudo apt-get install openjdk-8-jdk
```

### Step 3: Download and unzip the Hadoop tarball

Download Hadoop 3.3.1 from the Apache Hadoop releases page. For this exercise we will use the binary tarball.

```
$ wget https://www.apache.org/dyn/closer.cgi/hadoop/common/hadoop-3.3.1/hadoop-3.3.1.tar.gz
```

Create a directory with the name `hadoop` under your home folder. Unzip the binary package at the `hadoop` directory and then move to the `hadoop` directory.

```
$ mkdir hadoop
```

```
$ tar -xvzf hadoop-3.3.1.tar.gz -C ~/hadoop
```

```
$ cd hadoop-3.3.1
```

### Step 4: Configure passphraseless ssh

First, ensure that the SSH to localhost is operating.

```
$ssh localhost
```

```
Welcome to Ubuntu 16.04.7 LTS (GNU/Linux 4.15.0-142-generic i686)
```

\* Documentation: <https://help.ubuntu.com>

\* Management: <https://landscape.canonical.com>

\* Support: <https://ubuntu.com/advantage>

```
14 packages can be updated.
```

```
2 updates are security updates.
```

```
Last login: Fri Oct 29 00:34:32 2021 from 127.0.0.1
```

In the case that you are not able establish a SSH connection to the localhost without a passphrase, run the following command to initialize your private and public keys.

```
$ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa
```

```
$cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```



```
$chmod 0600 ~/.ssh/authorized_keys
```

### Step 5: Configure the pseudo-distributed mode (Single-node mode)

This includes the following steps:

#### Step 5.1: Setup environment variables

This is an optional step. Setup environment variables by adding the following lines at the file `~/.bashrc`. Ensure that you have defined the correct paths of the `JAVA_HOME` and `HADOOP_HOME` variables.

```
export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-i386
```

```
export HADOOP_HOME=~/.hadoop/hadoop-3.3.1
```

```
export PATH=$PATH:$HADOOP_HOME/bin
```

```
export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
```

Execute the following command to source the latest variables.

```
$source ~/.bashrc
```

To verify the value of a variable (e.g. `HADOOP_HOME`) run the following command.

```
$echo $HADOOP_HOME
```

#### Step 5.2: Edit the hadoop configuration files

Each Hadoop's component is configured using an XML file. The common properties of Hadoop are configured at the file `core-site.xml`, while the properties of HDFS, MapReduce, and YARN are configured at the files, `hdfs-site.xml`, `mapred-site.xml`, and `yarn-site.xml` respectively. These configuration files are all located in the directory `etc/hadoop` of Hadoop installation.

a) Edit the file `hadoop-env.sh`

```
$vi hadoop/hadoop-3.3.1/etc/hadoop/hadoop-env.sh
```

by setting a `JAVA_HOME` environment variable. Ensure that you define the correct edition of Open JDK installed at your system.

```
export JAVA_HOME=/usr/lib/jvm/java-penjdk-i386
```

b) Edit the file `core-site.xml`:

```
$vi hadoop/hadoop-3.3.1/etc/hadoop/core-site.xml
```

by adding the following configuration

```
<configuration>
```

```
<property>
```

```
<name>fs.defaultFS</name>
```

```
<value>hdfs://localhost:9000</value>
```

```
</property>
```

```
</configuration>
```

Optionally, you can configure the HDFS locations at the same configuration file.

```
<property>
```

```
<name>dfs.namenode.name.dir</name>
```

```
<value>/data/dfs/namespace_logs_330</value>
```

```
</property>
```

```
<property>
```

```
<name>dfs.datanode.data.dir</name>
```

```
<value>/data/dfs/data_330</value>
```

```
</property>
```

c) Edit the file `hdfs-site.xml`, i.e. the HDFS configuration file

```
$vi hadoop/hadoop-3.3.1/etc/hadoop/hdfs-site.xml
```

by adding the following configuration:

```
<configuration>
```

```
<property>
```

```
<name>dfs.replication</name>
```

```
<value>1</value>
```

```
</property>
```

```
</configuration>
```

d) Edit the file `mapred-site.xml`, i.e. the MapReduce configuration site.

```
$vi hadoop/hadoop-3.3.1/etc/hadoop/mapred-site.xml
```

by adding the following configuration:

```
<configuration>
```

```
<property>
```

```
<name>mapreduce.framework.name</name>
```

```
<value>yarn</value>
```

```
</property>
```

```
<property>
```

```
<name>mapreduce.application.classpath</name>
```

```
<value>${HADOOP_MAPRED_HOME}/share/hadoop/mapreduce/*:${HADOOP_MAPRED_HOME}/share/hadoop/mapreduce/lib/*</value>
```

```
</property>
```

```
</configuration>
```

e) Edit the file `yarn-site.xml`, i.e. the YARN configuration file.

```
$vi hadoop/hadoop-3.3.1/etc/hadoop/yarn-site.xml
```

by adding the following configuration

```
<configuration>
```

```
<property>
```

```
<name>yarn.nodemanager.aux-services</name>
```

```
<value>mapreduce_shuffle</value>
```

```
</property>
```

```
<property>
```

```
<name>yarn.nodemanager.env-whitelist</name>
```

```
<value>JAVA_HOME,HADOOP_COMMON_HOME,HADOOP_HDFS_HOME,HADOOP_CONF_DIR,CLASSPATH_PREPEND_DISTCACHE,HADOOP_YARN_</value>
```

```
</property>
```

```
</configuration>
```

**Step 5.3: Format NameNode**

Move to the bin folder of Hadoop installation (i.e. /hadoop/hadoop-3.3.1/bin) and format the NameNode

```
$hdfs namenode -format
```

**Step 5.4: Run the DFS and YARN daemons**

Move to the sbin folder of Hadoop installation (i.e. /hadoop/hadoop-3.3.1/sbin) that contains the scripts to start/stop the Hadoop daemons.

Run the following command to start the DFS daemons

```
$. /start-dfs.sh
```

Starting namenodes on [localhost]

Starting datanodes

Starting secondary namenodes [ubuntu]

By running the above script 3 daemons are started, one for the NameNodes, a second for the secondary NameNodes and a third for the DataNodes

At the same folder (sbin), run the following command to start the YARN daemons.

```
$. /start-yarn.sh
```

Starting resourcemanager

Starting nodemanagers

By running the above script, 2 daemons are started, one for the ResourceManager and a second for the NodeManagers.

**Step 5.5: Confirm that Hadoop has installed correctly**

To verify the running java process run the following command which is the ideal case.

```
$jps
```

5728 NameNode

6097 SecondaryNameNode

6482 NodeManager

5865 DataNode

6797 Jps

6349 ResourceManager

You can access the NameNode web portal from the following URL:

<http://localhost:9870/dfshealth.html#tab-overview>

and see the picture below.

**Overview** 'localhost:9000' (✓active)

<b>Started:</b>	Tue Nov 09 02:29:26 -0800 2021
<b>Version:</b>	3.3.1, rs399c37a397a54188041d80621bdefc46885f2
<b>Compiled:</b>	Mon Jun 14 22:13:00 -0700 2021 by ubuntu from (HEAD detached at release-3.3.1-RC3)
<b>Cluster ID:</b>	CID-acb0e5d9-12b6-4d0e-86c6-d28ce982c066
<b>Block Pool ID:</b>	BP-2123813449-127.0.1.1-1636453703536

**Summary**

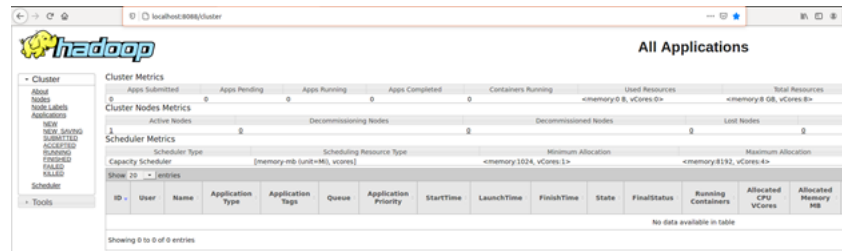
Security is off.  
 Safemode is off.  
 1 files and directories, 0 blocks (0 replicated blocks, 0 erasure coded block groups) = 1 total filesystem object(s).  
 Heap Memory used 97.16 MB of 219.75 MB Heap Memory. Max Heap Memory is 663.25 MB.  
 Non Heap Memory used 30.41 MB of 31.48 MB Committed Non Heap Memory. Max Non Heap Memory is «unbounded».

<b>Configured Capacity:</b>	18.62 GB
<b>Configured Remote Capacity:</b>	0 B
<b>DFS Used:</b>	24 KB (0%)
<b>Non DFS Used:</b>	6.91 GB
<b>DFS Remaining:</b>	10.74 GB (57.7%)
<b>Block Pool Used:</b>	24 KB (0%)

You can also access the the YARN web portal from the following URL:

<http://localhost:8088/cluster>

and see the picture below.



# The Hadoop Distributed File System

Site: [DTAM Online Training Platform](#)  
Course: Big Data  
Book: The Hadoop Distributed File System

Printed by: Vasileios Gkamas  
Date: Friday, 27 October 2023, 4:21 PM

## Description

## Table of contents

1. HDFS Architecture
2. Block replication in Hadoop
3. Rack Awareness
4. Hadoop HDFS Operations
5. Interacting with HDFS

# 1. HDFS Architecture

Hadoop is optimized to store a small number of large files rather than a big number of small files. HDFS is a high reliability storage solution preventing data loss even in the case of hardware failure. Moreover, it provides high throughput access to application data by providing parallel data access.

HDFS follows the master-slave architecture of Hadoop; it has two types of nodes that work in the same manner:

- **HDFS Master (NameNode)** manages file access to the clients. It maintains and manages the slave nodes and assigns tasks to them. It executes file system namespace operations like opening, closing, and renaming files and directories. It also determines the mapping of blocks to DataNodes. NameNode stores metadata like filename, the number of blocks, number of replicas, a location of blocks, block IDs, etc. The NameNode keeps a reference to every file and block in the filesystem in memory, which means that on very large clusters with many files, memory becomes the limiting factor for scaling. HDFS federation, introduced in the 2.x release series, allows a cluster to scale by adding NameNodes, each of which manages a portion of the filesystem namespace.
- **HDFS Slave (DataNodes)**. These are the actual worker nodes that do the tasks and serve read and write requests from HDFS's clients. DataNodes are arranged in racks. They perform block-related tasks, such as block creation, deletion, and replication upon request from the NameNode. Once a block is written on a DataNode, it is replicated to other DataNodes, and this process continues until creating the required number of copies.

NameNode and DataNode each run an internal web server to display basic information about the current status of the cluster. With the default configuration, the NameNode front page is at <http://namenode-name:9870/>. It lists the DataNodes in the cluster and basic statistics of the cluster. The web interface can also be used to browse the file system.

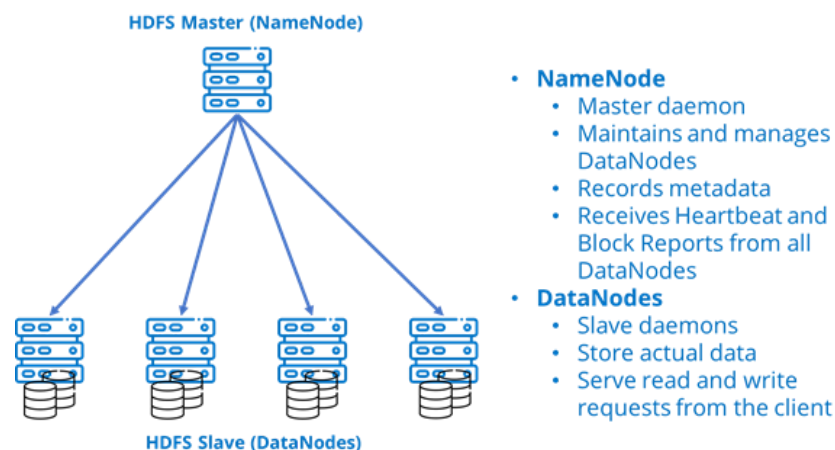


Figure 3.5: HDFS NameNode and DataNodes

The communication protocols in HDFS are running over the TCP/IP protocol. Using the ClientProtocol protocol, a client establishes a connection to a TCP port of the NameNode, while using the DataNode protocol, the DataNodes talk to the NameNode. A Remote Procedure Call (RPC) abstraction wraps both the Client Protocol and the DataNode Protocol. By design, the NameNode never initiates any RPCs. Instead, it only responds to RPC requests issued by DataNodes or clients.

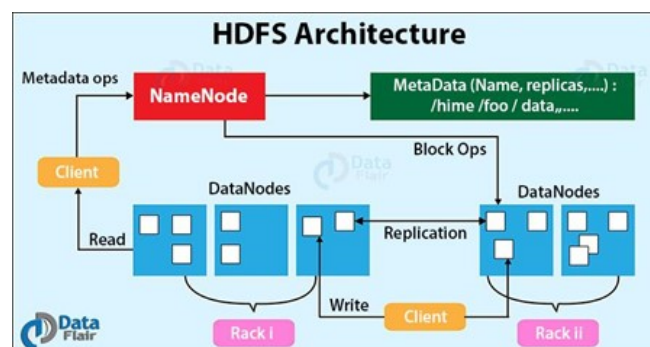


Figure 3.6: The HDFS architecture (source: <https://data-flair.training/blogs/hadoop-hdfs-architecture/>)



## 2. Block replication in Hadoop

HDFS splits the files into small pieces of data known as blocks with 128 MB default size. The blocks are stored in a Hadoop cluster on different nodes in a distributed manner. This provides a mechanism for MapReduce to process the data in parallel in the cluster. In the case that the data size is less than the block size, the block size will be equal to the data size. For example, if the file size is 140 MB, then 2 blocks will be created, one with the default size 128 MB and another with 12 MB size.

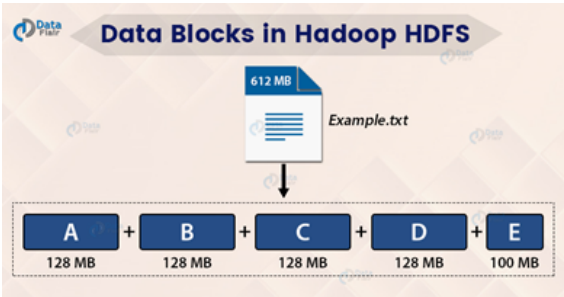


Figure 3.7: Data blocks in HDFS (source: <https://data-flair.training/blogs/hadoop-hdfs-architecture/>)

HDFS replicates the blocks of a file on different nodes of the cluster for fault tolerance. By default, the HDFS replication factor is 3. However, both the block size and replication factor are configurable per file. The NameNode is responsible for managing the replication of blocks. The NameNode periodically receives a Heartbeat and a Blockreport from each of the DataNodes in the cluster. The receipt of a Heartbeat implies that the DataNode is functioning properly, while a Blockreport contains a list of all blocks stored in a DataNode.

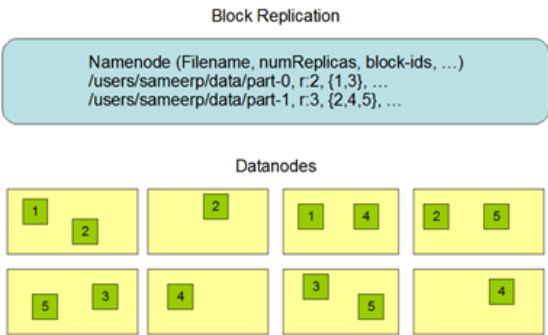


Figure 3.8: Block replication in HDFS (source: <https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html>)

### 3. Rack Awareness

HDFS stores files (in fact blocks) across multiple DataNodes in a Hadoop cluster. HDFS has the feature of "Rack Awareness", meaning that it chooses the DataNodes on the same rack or near racks for file read/write in order to get the maximum performance and improve the network traffic during the execution of data read/write operations.

In a large Hadoop cluster exist many racks, each of which hosts a set of DataNodes. A rack is a collection of around 40-50 DataNodes that are reconnected using a switch. The NameNode maintains rack IDs of each DataNode. If the switch goes down, all the DataNodes of the rack will become unavailable. It is obvious, that in terms of optimization of network traffic during the execution of file read/write requests, the communication between the DataNodes on the same rack is more efficient in comparison with the communication between DataNodes residing on different racks, as, the network bandwidth between nodes within the same rack is higher than the network bandwidth between nodes on a different rack.

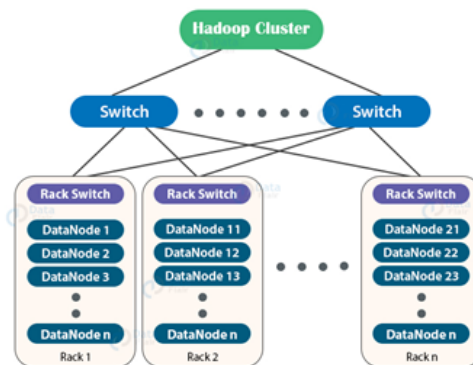


Figure 3.9: Rack Awareness in HDFS (source: <https://data-flair.training/blogs/rack-awareness-hadoop-hdfs/>)

The Rack Awareness benefits are:

- Reduced network traffic and latency during the execution of file read/write functions, thus improving the overall performance of the cluster. Moreover, the YARN can optimize the MapReduce job performance, by assigning tasks to nodes that are 'closer' to their data in terms of network topology.
- Improved fault tolerance. Given that the block replicas are stored at different racks, no data loss occurs even if the rack fails
- Improved data availability. The data is available even in unfavorable conditions.
- Minimize the writing cost and maximize read speed. The Rack awareness policy places read/write requests to replicas which are in the same rack. This minimizes writing cost and maximizes read speed.

One question that is raised at Rack Awareness is the location of data block replicas. If we store replicas on different DataNodes on the same rack, then the network bandwidth is improved. However, if the rack fails (something that rarely happens), then there will be no copy of blocks on another rack. On the other hand, by placing block replicas on DataNodes located at different racks, the fault tolerance is improved, but the network bandwidth among the DataNodes storing the block replicas is degraded.

To address this issue, the NameNodes of a cluster that contains many racks, maintain the block replication by using the following Rack awareness policies:

- A DataNode should not store more than one replica.
- On the same rack should not be placed more than two replicas.
- The number of racks used for block replication should be smaller than the number of replicas.

In the case of the default replication factor, the 3 block replicas are stored as follows:

- The first replica is stored at a DataNode on the local rack.
- The second replica is stored on a different DataNode on the same rack.
- The third replica is stored on a DataNode at a different rack.

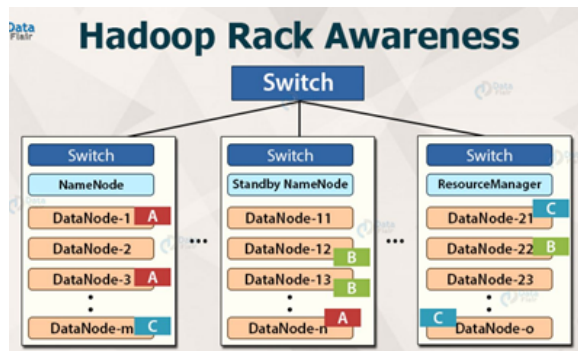


Figure 3.10: Example of block replication in HDFS considering Rack Awareness policies (source: <https://data-flair.training/blogs/rack-awareness-hadoop-hdfs/>)

Moreover, in the case of copying a block replica, the second replica is placed on a DataNode on a different rack, if one replica exists. If the existing replicas are two and are on the same rack, then place the third replica on a DataNode located at a different rack. If the replication factor is greater than 3, the placement of the fourth and following replicas are determined randomly while keeping the number of replicas per rack below the upper limit (which is basically  $(\text{replicas} - 1) / \text{racks} + 2$ ). Because the NameNode does not allow DataNodes to have multiple replicas of the same block, the maximum number of replicas created is the total number of DataNodes at that time.

Considering the aforementioned rules, Figure 3.10 depicts how the 3 replicas of 3 data blocks (A, B and C) are stored in 3 racks.

## 4. Hadoop HDFS Operations

HDFS has many similarities with the Linux file system, so we can do almost all the operations that are available at a local file system, such as create a directory, copy file or directory, move a file, change permissions, etc. HDFS also provides different access rights to users, groups of users and others.

### 1. Read Operation

When a client wants to execute a read operation (i.e. read a file), it first interacts with the NameNode as it is the one which stores the metadata.

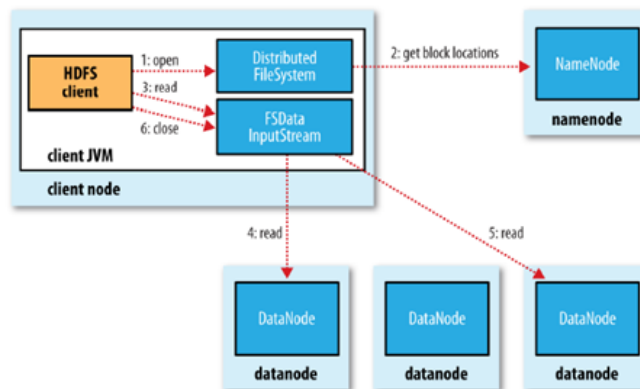


Figure 3.11: HDFS read operation (source: Hadoop the Definitive Guide, O'Reilly Media, Inc)

The process is the following:

- The HDFS client opens the file it wishes to read by calling the `open()` method on the `FileSystem` object, which for HDFS is an instance of `DistributedFileSystem`.
- The `DistributedFileSystem` sends a request to the `NameNode` to get block locations.
- The `NameNode` checks if the client has the required privileges to access the data.
- If the client has the required privileges, the `NameNode` specifies the address of the slave nodes (`DataNodes`) where the data is stored (then the client interacts with the specific `DataNodes` and reads the data).
- The `NameNode` provides the HDFS client with a token that is checked by the `DataNode` for reading the file.
- After this check, the `DataNode` allows the client to read the specific block.
- Finally, the HDFS client opens the input stream and starts reading the data from the identified `DataNode`.

### 2. Writing Operation

Initially the client creates the file by calling the `create()` method on the `DistributedFileSystem`. Then the client interacts with the `NameNode` which provides the address of the `DataNode` on which the data must be written. Once the writing operation is completed, the `DataNode` starts replicating the block into another `DataNode` and a third `DataNode`. Moreover, once the required replication is completed, an acknowledgment is sent to the client. The authentication is the same as that in the read operation.

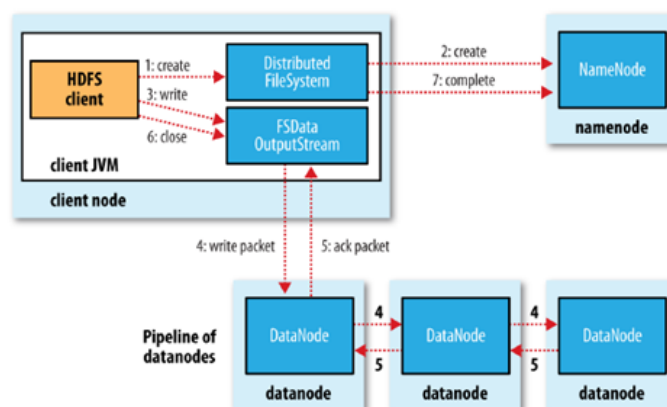


Figure 3.12: HDFS write operation (source: Hadoop the Definitive Guide, O'Reilly Media, Inc)

## 5. Interacting with HDFS

This subsection presents how to interact with HDFS using the built-in commands. The current working directory is the HDFS home directory `/user/<username>` that often has to be created manually. The HDFS home directory can also be implicitly accessed, e.g., when using the HDFS trash folder, the `.Trash` directory in the home directory.

The interaction of the end-user with HDFS is done from the command line interface using the **hdfs** script by the following way.

**\$ hdfs COMMAND [-option <arg>]**

The script uses two arguments:

- (a) the **COMMAND** argument that defines the HDFS functionality to be used and
- (b) the **-option** argument that is the name of a specific option for the specified command, while **<arg>** is the argument(s) that are identified for the specific option.

Note that all HDFS commands are invoked by the `bin/hdfs` script. Running the `hdfs` script without any arguments prints the description for all commands.

In order to perform basic file manipulation operations on HDFS, we use the `hdfs` script with the **dfs command** as an argument. This command supports many operations of files that are also found in the Linux shell. Like Linux, the `hdfs` command runs with the permission of the system user running the command. In the following examples, we assume that the user running the scripts is named "hadoop".

Below are presented some common commands for files manipulation:

### List the contents of a directory.

To list the contents of a directory, use the following command

**\$ hdfs dfs -ls [-R] <args>**

If we run the `-ls` command without any arguments on a new cluster, it will not return any results, as the command will display the contents of the user's home directory on HDFS (not on the host machine) which by default do not exist and should be created.

The `-R` option will return stat recursively through the directory structure.

In the case that we run the `-ls` command using an `"/"` argument, the contents of the root folder of HDFS will be displayed. For example:

**\$ hdfs dfs -ls /**

**Found 2 items**

**drwxr-xr-x - hadoop supergroup 0 2020-10-20 14:36 /hadoop**

**drwx----- - hadoop supergroup 0 2020-10-20 14:36 /tmp**

You will observe that the output of this command is the same with the output of the `'ls'` command running on a Unix/Linux file system. The command displays the files and folders located under the HDFS root folder, as well as the permissions for owners and groups. The two folders displayed (`hadoop` and `tmp`) are automatically created when HDFS is formatted. The "hadoop" user is the name of the user under which the Hadoop daemons were started, while the "supergroup" is the group of the superusers in HDFS.

### Creating a directory.

To create a directory within HDFS you should execute the following command:

**\$ hadoop dfs -mkdir [-p] <paths>**

The `-p` option behavior is much like Unix, thus creating parent directories along the path.

The home directories of the users within HDFS are stored in the folder `/user/$HOME`. From the previous example, it is observed that the "user" directory does not exist under the root directory ("`/`"), so we have to create this directory using the following command.

**\$ hdfs dfs -mkdir /user**

Moreover, to create a home directory for the user "usrhd" the following command will be used.

```
$ hdfs dfs -mkdir /user/usrhd
```

### Deleting a directory

To delete a directory within HDFS the following command can be used

```
$ hdfs dfs -rmdir <directory>
```

For example, the following command deletes the directory "emptydir" located at the home path of "usrhd" user, i.e. the path /usr/usrhd.

```
$ hdfs dfs -rmdir /user/usrhd/emptydir
```

### Copying data in a directory

To copy data in a HDFS directory use the following command

```
$ hdfs dfs -put <localfile> <full_path_to_HDFSdirectory>
```

For example, to copy at the HDFS directory /user/usrhd a file named "myfile.txt" that is located at the local file system under the path /home/hadoop/myfile.txt, the following command is used.

```
$ hdfs dfs -put /home/hadoop/myfile.txt /user/usrhd/myfile.txt
```

The -put command can run with the following options:

-p : Preserves access and modification times, ownership and the permissions. (assuming the permissions can be propagated across filesystems)

-f : Overwrites the destination if it already exists.

-t <thread count> : Number of threads to be used, default is 1. Useful when uploading a directory containing more than 1 file.

-l : Allow DataNode to lazily persist the file to disk, Forces a replication factor of 1. This flag will result in reduced durability. Use with care.

-d : Skip creation of temporary file with the suffix \_COPYING\_.

Note that if you want to copy data in a different direction, i.e. from HDFS to the local file system the -get command should be used as follows.

```
$ hdfs dfs -get /user/usrhd/myfile.txt /home/hadoop
```

### Changing the permission of a file

To change the read/write/execute permission of a file use the following command

```
$ hadoop fs -chmod [-R] filename
```

For example, the following command changes the permission of the file named "logs.csv" to 744.

```
$ hdfs dfs -chmod 744 /data/logs.csv
```

With -R, make the change recursively through the directory structure. The user must be the owner of the file, or else a super-user.

Take into account that the aforementioned commands are basic ones for file operations in HDFS. For a complete list of file manipulation commands possible with hdfs dfs please refer to this [link](#). For help with a specific option, use either **hdfs dfs -usage <option>** or **hdfs dfs -help <option>**.

### Show the capacity, free and used space of the filesystem

The following command shows the capacity, free and used space of a specific directory.

```
$ hdfs dfs -df /user/usrhd/dir1
```

The -h option will format file sizes in a "human-readable" fashion.

```
$ hdfs dfs -df [-h] <directory>
```

The following command shows the size of files and directories contained in the given directory or the length of a file in case it is just a file.

```
$ hdfs dfs -du [-s] [-h] [-v] [-x] <directory>
```

For example:

```
$ hdfs dfs -du /user/usrhd/dir1
```

The different options of the command are the following:

- The -s option will result in an aggregate summary of file lengths being displayed, rather than the individual files.
- The -h option will format file sizes in a "human-readable" fashion (e.g. 64.0m instead of 67108864)
- The -v option will display the names of columns as a header line.
- The -x option will exclude snapshots from the result calculation. Without the -x option (default), the result is always calculated from all Nodes, including all snapshots under the given path.

## Exercise: Interacting with Hadoop running at Docker containers

Site: [DTAM Online Training Platform](#)

Course: Big Data

Book: Exercise: Interacting with Hadoop running at Docker containers

Printed by: Vasileios Gkamas

Date: Friday, 27 October 2023, 4:21 PM



## Table of contents

1. General description
2. Desired objectives
3. Required material
4. Other requirements
5. Solution

# 1. General description

This exercise concerns HDFS. One can regard HDFS as a regular file system, however in fact many HDFS shell commands are inherited from the corresponding bash commands. At this example the different Hadoop daemons (e.g. resource manager and namenode) are running at docker containers.

To run a file system command on HDFS use the command

`hdfs COMMAND [-options <arg>]`

Note that one can use interchangeably **hadoop fs** or **hdfs dfs** when working on a HDFS file system. The command `hadoop` is more generic because it can be used not only on HDFS but also on other file systems that Hadoop supports (such as Local FS, WebHDFS and others).

To get more help on a specific `hdfs` command use: **hdfs dfs -help <command>**.

For example

```
vgkamas@ubuntu:~$ hdfs dfs -help rm
```

```
-rm [-f] [-r|-R] [-skipTrash] [-safely] <src> ... :
```

Delete all files that match the specified file pattern. Equivalent to the Unix command "`rm <src>`"

- `-f` If the file does not exist, do not display a diagnostic message or modify the exit status to reflect an error.
- `[-rR]` Recursively deletes directories.
- `-skipTrash` option bypasses trash, if enabled, and immediately deletes `<src>`.
- `-safely` option requires safety confirmation, if enabled, requires confirmation before deleting large directory with more than `<hadoop.shell.delete.limit.num.files>` files. Delay is expected when walking over large directory recursively to count the number of files to be deleted before the confirmation.

## 2. Desired objectives

The desired objectives of the exercise are the following:

- Get familiar with Docker containers
- Get familiar with the HDFS

### 3. Required material

For the execution of this exercise you need access at a Hadoop installation running with Docker containers

Other useful material

- Towards Data Science, [HDFS Simple Docker Installation Guide for Data Science Workflow](#)

## 4. Other requirements

There are no other special requirements for the execution of this exercise.

## 5. Solution

Initially make an SSH connection to the Hadoop cluster (192.168.50.70:9870) and execute the following steps.

List the running containers

```
$ docker container ls
```

You should see an output like the following one:

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
541197d0b2be	bde2020/hadoop-historyserver:2.0.0-hadoop3.2.1-java8	"/entrypoint.sh /run..."	13 days ago	Up 13 days (healthy)	8188/tcp
328dec436b4a	bde2020/hadoop-nodemanager:2.0.0-hadoop3.2.1-java8	"/entrypoint.sh /run..."	13 days ago	Up 13 days (healthy)	8042/tcp
b79c8336c7f4	bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8	"/entrypoint.sh /run..."	13 days ago	Up 13 days (healthy)	0.0.0.0:9000->9000/tcp, :::9000->9000/tcp, 0.0.0.0:9870->9870/tcp, :::9870->9870/tcp
870/tcp	namenode				
175c28a39a1a	bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8	"/entrypoint.sh /run..."	13 days ago	Up 13 days (healthy)	9864/tcp
	datanode				
9cc4f01b5bcb	bde2020/hadoop-resourcemanager:2.0.0-hadoop3.2.1-java8	"/entrypoint.sh /run..."	13 days ago	Up 13 days (healthy)	8088/tcp
	resourcemanager				

Connect at the container running the namenode daemon

```
$ docker exec -it namenode /bin/bash
```

From this container you will run all commands

In the case that you want to copy a file from the local machine to the container, use the following command

```
$ docker cp myfile.txt ccae4670f030:/hadoop
```

For example, the above command copies the file 'myfile.txt' from the local machine at the /hadoop folder of the container with ID ccae4670f030.

### Part 1: Creating and removing directories

1.1 Create a directory named 'user' under the root directory of HDFS.

```
$ hdfs dfs -mkdir /user
```

1.2 Create a directory named 'temp' under the root directory of HDFS

```
$ hdfs dfs -mkdir /temp
```

1.3 Create a directory named 'tmp' under the root directory of HDFS

```
$ hdfs dfs -mkdir /tmp
```

1.4 Remove the 'tmp' HDFS directory

```
$ hdfs dfs -rmdir /tmp
```

1.5 Create a directory named 'exercise2' inside the 'user' directory.

```
$ hdfs dfs -mkdir /user/exercise2
```

### Part 2: Coping files from/to HDFS

2.1 Create at your home directory in the local machine a file named 'test1.txt' that contains the text "hello world hello everybody".

```
$ echo 'hello world hello everybody' > test1.txt
```

2.2 Copy the file test1.txt from the local machine to the HDFS directory /user/exercise2.

```
$ hdfs dfs -put test1.txt /user/exercise2
```

Using the -p argument preserves the access and modification times, the ownership and the mode of the file. If you will try to copy again the same file at the same location, you will see a message saying that the file already exists. To overwrite the file, use the put command with the -f option.

```
$ hdfs dfs -put -f test1.txt /user/exercise2
```

If you want to move the file from the local machine to HDFS, then you must use the command 'hdfs dfs -moveFromLocal'.

2.3 Copy the file test1.txt from HDFS back to the local machine. At the following command you should replace the path '/home/vgkamas' with the actual path at your machine.

```
$ hdfs dfs -get -f /user/exercise2/test1.txt /home/vgkamas
```

Using the -p argument preserves the access and modification times, the ownership and the mode of the file. The -f option overwrites the destination file if it already exist (like in our case).

### Part 3: Reading/writing HDFS files

3.1 List the contents of the file test1.txt that is stored in HDFS

```
$ hdfs dfs -cat /user/exercise2/test1.txt
```

```
hello world hello everybody
```

3.2 Create at your home directory of the local machine a file named 'test2.txt' that contains the text "hadoop processing distributed file system".

```
$ echo 'hadoop processing distributed file system ' > test2.txt
```

3.3 Append the contents of the local file test2.txt at the HDFS file test1.txt

```
$ hdfs dfs -appendToFile test2.txt /user/exercise2/test1.txt
```

3.4 Show the contents of the HDFS file test1.txt

```
$ hdfs dfs -cat /user/exercise2/test1.txt
```

```
hello world hello everybody
```

```
hadoop processing distributed file system
```

### Part 4: Listing files and directories

4.1 List all the files/directories under the HDFS root directory.

```
$ hdfs dfs -ls /
```

4.2 List all the files/directories under the HDFS directory /user

```
$ hdfs dfs -ls /user
```

4.3 Recursively list all files and directories under the HDFS root directory.

```
$ hdfs dfs -ls -R /
```

### Part 5: File management

5.1 Copy the HDFS file test1.txt from the folder /user/exercise2/ to the HDFS folder /temp

```
$ hdfs dfs -cp /user/exercise2/test1.txt /temp
```

The -p option preserves the access and modification times, the ownership and the mode of the copied file, while the -f option overwrites the destination if it already exists.

5.2 Delete the file test1.txt from the HDFS directory /user/exercise2

```
$ hdfs dfs -rm /user/exercise2/test1.txt
```

The -R option deletes the directory and any content under it recursively.

5.3 Move the file test1.txt from the HDFS directory /temp to the HDFS directory /user

```
$ hdfs dfs -mv /temp/test1.txt /user
```

**Part 6: Checking the size of filesystem, directories and files**

6.1 Show the capacity, free and used space of the HDFS directory /user.

```
$ hdfs dfs -df -h /user
```

6.2 Show the amount of space, in bytes, used by the files /user/test1.txt

```
$ hdfs dfs -du -h /user/test1.txt
```

Take into account that in both commands ('hdfs dfs -h' and 'hdfs dfs -du') the -h option is optional and is used in order to format the sizes of files in a human-readable fashion.



# The Hadoop Yarn

Site: [DTAM Online Training Platform](#)  
Course: Big Data  
Book: The Hadoop Yarn

Printed by: Vasileios Gkamas  
Date: Friday, 27 October 2023, 4:21 PM

## Table of contents

1. Hadoop Yarn architecture
2. Resource Manager
3. Node Manager
4. Application Master
5. Interacting with YARN

# 1. Hadoop Yarn architecture

Apache **YARN (Yet Another Resource Negotiator)** is Hadoop's resource management layer that was introduced in Hadoop 2.x. Yarn provides two core services, job scheduling and resource management. It supports various data processing engines such as graph processing, interactive processing, stream processing and batch processing to run and process data stored in HDFS.

The core idea of YARN is to divide the functionalities of resource management and job scheduling/monitoring into separate daemons. Like HDFS, Hadoop YARN follows a master-slave architecture. It consists of a master daemon known as **Resource Manager (RM)**, slave daemons called **Node Manager** (one per slave node) and **Application Master (AM)** (one per application). An application is either a single job or a Directed Acyclic Graph (DAG) of jobs.

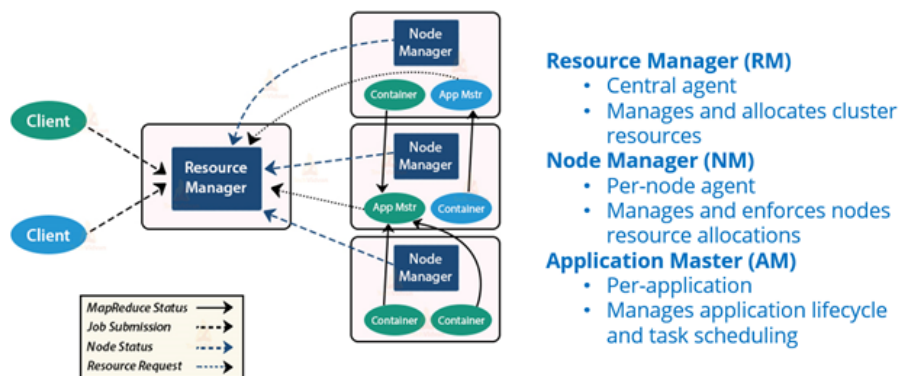


Figure 3.13: The Hadoop Yarn architecture (source: <https://techvidvan.com/tutorials/how-hadoop-works-internally/>)

In order to understand the architecture of YARN, it is important to get familiarized with the following terms:

- A **client** is the program that submits YARN jobs to the cluster. Sometimes a client also refers to the gateway machine on which the client program runs.
- A **job**, also called an application (for example, a MapReduce job will include mappers, optional reducers and the list of inputs they process), contains one or more tasks.
- When running a MapReduce job, a **task** can be either a mapper or a reducer task. There are some applications that use both mappers and reducers and some that just use mappers and no reducers. Each mapper and reducer task runs within its own Container. The administrator configures the size of the Containers. The job determines the number of mappers and reducers.
- A **Container** is a collection of physical resources such as RAM, CPU cores and disk on a single node. For example, a Container can represent 4GB memory and 2 processing cores. All the YARN application tasks run in Containers. Each Hadoop job contains multiple tasks and each of the tasks runs in its own container. A Container comes into life when the task starts. When the task completes, the Container is terminated, and its resources are allocated to other tasks.

To run an application to YARN, the client contacts the Resource Manager and asks it to run an Application Master process. The Resource Manager then finds a Node Manager that can launch the Application Master in a Container.

What the Application Master does once it is running depends on the application. In one scenario, the Application Master could run a computation in the Container it is running in and return the result of the computation to the client. In another scenario, the Application Master could request more Containers from the Resource Manager and use them to run a distributed computation.

YARN has flexibility for making resource requests. A request for a set of containers expresses the amount of computer resources (i.e. CPU and memory) required for each Container, as well as locality constraints for the containers. This is critical in order to ensure that the distributed data processing algorithms use the bandwidth of the cluster efficiently.

The application workflow in Hadoop YARN is the following:

- The client submits an application to the Resource Manager.
- The Resource Manager allocates a Container to start the Application Manager.
- The Application Manager registers itself with the Resource Manager.
- The Application Manager negotiates Containers from the Resource Manager.
- The Application Manager notifies the Node Manager to launch Containers.
- The code of the application is executed in the Container.
- The client contacts the Resource Manager/Application Manager to monitor the status of the application.
- Once the processing is complete, the Application Manager un-registers from the Resource Manager.



## 2. Resource Manager

The Resource Manager (RM) is the master daemon of YARN. This daemon manages the assignment of resources, i.e. CPU and memory among the applications.

The core features of the RM are the following:

- It creates the first container for an application. This is the container in which will run the ApplicationMaster for the application.
- It tracks the heartbeats from the NodeManagers to manage the DataNodes.
- It runs a Scheduler to manage the allocation of resources among the nodes of the clusters.
- It manages security at cluster level.
- It manages the resource requests coming from the ApplicationMasters.
- It monitors the status of the ApplicationMaster and restarts its Container upon its failure.
- It deallocates the Containers when the application completes or after they expire.

The RM has two main components:

- **The Scheduler.** The Scheduler is specifically designed to negotiate resources and not schedule tasks. The Scheduler is responsible for resources allocation to the running applications. The scheduler does not perform application's monitoring or tracking and does not provide any guarantee about restarting failed tasks either due hardware or application failure. The scheduling is performed based on the resource requirements of the applications, based on the abstract notion of a resource Container which incorporates elements such as CPU, memory, disk, network etc. The Scheduler has a pluggable policy plug-in, which is responsible for partitioning the cluster resources among the various queues, applications etc. The Applications can request resources at different layers of the cluster topology such as nodes, racks etc. Hence, the Scheduler determines how much and where to allocate resources based on the availability of resources and the configured sharing policy.
- **The Application Manager.** The Application Manager is responsible for maintaining a collection of submitted applications. It accepts a job from the client and negotiates for a container to execute the application-specific Application Master. More specifically, it is responsible for accepting job submissions, negotiating the first container for executing the application-specific Application Master and provides a service for restarting the Application Master container on failure.

The RM works together with the per-node Node Managers and the per-application Application Masters.

The components that interface the RM to the client are two:

- **ClientService.** This is the client's interface to the RM. It manages all the RPC interfaces from the clients to the RM including operations like application submission and termination, obtaining queue information, obtaining cluster statistics, etc.
- **AdminService.** This interface serves all the admin operations like refreshing the list of nodes, configuring the queues, etc. This is necessary in order to ensure that the admin requests don't get starved due to the normal users' requests and to give a high priority to the execution of admin commands.

The components that connect the RM to the nodes are three:

- **ResourceTrackerService.** This component obtains heartbeats from nodes in the cluster and forwards them to the YarnScheduler. It responds to RPCs from all nodes, registers new nodes and rejects requests from any invalid/decommissioned nodes.
- **NodesListManager.** This component manages valid and decommissioned nodes. It is responsible for reading the host configuration files and seeding the initial list of nodes based on those files. It also keeps track of decommissioned nodes.
- **NMLivelinessMonitor.** This component keeps track of live nodes and dead nodes based on their last heartbeat time. Any node that doesn't send a heartbeat within a configured interval of time (its default value is 10 minutes), is considered dead and is expired by the RM. All the containers currently running on an expired node are marked as dead and no new containers are scheduled on such nodes.

The components that are interacting with the per-application Application Masters are two:

- **ApplicationMasterService.** This component manages all the RPCs from the AMs, such as registration of new AMs and termination/unregister-requests from any finishing AMs and forward them to the YarnScheduler.
- **AMLivelinessMonitor.** This component maintains the list of live AMs and dead/non-responding AMs using the heartbeats and registers and de-registers the AMs from the Resource manager. Hence, all the Containers currently running/allocated to an expired AM are marked as dead. The ApplicationMasterService and AMLivelinessMonitor work together to maintain the fault tolerance of Application Masters.

### 3. Node Manager

The NM is the slave daemon of YARN. It is responsible for launching and managing containers on a node. Containers execute tasks as specified by the Application Master. The NM is also responsible for monitoring the containers' resource usage and reporting this to the RM, as well as, for tracking the health of the node on which it is running.

The core features of the NM are the following:

- It interacts with the ResourceManager through the heartbeats and Container status notifications.
- It registers and starts the application processes.
- It launches the ApplicationMaster and the rest of an application's containers (e.g., the map and reduce tasks that run in the Containers) on request from the ApplicationsManager.
- It oversees the lifecycle of the application Containers and it monitors, manages and provides information regarding the resource consumption by the Containers.
- It tracks the health of the DataNodes.
- It handles log management by aggregating the job logs and saving them to HDFS.
- It maintains security at the node level.

The separate components of the YARN's Node Manager are the following:

- **NodeStatusUpdater.** This component registers with the Resource Manager and sends information about the resources available to every node. This component also sends updates on the status of the Containers running at. The RM may signal the NodeStatusUpdater to potentially kill already running Containers.
- **Container Manager.** This is the core component of the Node Manager. It is responsible for managing the containers running on each node. It has several sub-components (RPC server, ResourceLocalizationService, ContainersLauncher, AuxServices, ContainersMonitor and LogHandler), each of which performs a functionality that is necessary for the management of the Containers running on the node.
- **Container Executor.** This component interacts with the underlying operating system to securely place files and directories needed by Containers and subsequently to securely launch and stop processes that are corresponding to Containers.
- **NodeHealthChecker Service.** This component checks the health of the node by periodically running a configured script. It also monitors the health of the disks, by creating temporary files on them. The health of the node is notified to the NodeStatusUpdater component which forwards this information to the RM.
- **ContainerTokenSecretManager:** This component examines the incoming requests for Containers ensuring that they are properly authorized by the RM.
- **WebServer.** This component exposes the list of applications, the Container's information running on the node at a given point of time, node's health information and the logs produced by the Containers.

## 4. Application Master

The Application Master coordinates the execution of an application in the cluster. There is a one-to-one mapping between each application and an AM. The AM negotiates resources, i.e. Containers from the RM and works with the NMs to execute and monitor the submitted jobs (MapReduce programs).

Once the execution of an AM starts as a Container, the AM periodically sends heartbeats to the RM to report its health and update the record of its resource demands. In fact, after building a model of its requirements, the AM encodes its preferences and constraints in a heartbeat message to the RM. The RM responds to the heartbeat message by sending to the Application Master, a lease of Containers that is bound to an allocation of resources at a particular node in the cluster. As the application progresses the allocation and deallocation of Containers is dynamically updated.

Consequently, one Application Master runs per application. It negotiates resources from the RM and works with the NM. The AM acquires Containers from the Scheduler of the RM before contacting the corresponding NMs to start the application's individual tasks.

## 5. Interacting with YARN

The YARN commands are invoked by the bin/yarn script. Running the yarn script without any arguments prints the description for all commands.

The syntax of the YARN commands is the following.

**\$ yarn [SHELL\_OPTIONS] COMMAND [GENERIC\_OPTIONS] [SUB\_COMMAND]**

where:

- SHELL\_OPTIONS is the common set of [shell options](#).
- GENERIC\_OPTIONS is the common set of options supported by [multiple commands](#).
- COMMAND COMMAND\_OPTIONS Various commands with their options.

Below are presented the most common YARN commands.

The following command shows the class path needed to get the Hadoop jar and the required libraries. If called without arguments, then prints the classpath set up by the command scripts, which is likely to contain wildcards in the classpath entries.

**\$ yarn classpath [--glob | --jar <path> | -h | --help]**

The following command is used to show and kill the Hadoop applications.

**\$ yarn application [options]**

For the complete list of options please refer to this [link](#).

The following command is used to show the applicationattempt.

**\$ yarn applicationattempt [options]**

For the complete list of options please refer to this [link](#).

The following command is used to show the container information.

**\$ yarn container [options]**

For the complete list of options please refer to this [link](#).

The following command is used to show the node information.

**\$ yarn node [options]**

For the complete list of options please refer to this [link](#).

The following command prints queue information.

**\$ yarn queue [options]**

For the complete list of options please refer to this [link](#).



# The Hadoop MapReduce

Site: [DTAM Online Training Platform](#)  
Course: Big Data  
Book: The Hadoop MapReduce

Printed by: Vasileios Gkamas  
Date: Friday, 27 October 2023, 4:21 PM

## Table of contents

1. What is MapReduce
2. MapReduce job execution flow
3. MapReduce data processing example
4. How Hadoop runs a MapReduce job
5. Hadoop streaming
6. Running a simple MapReduce job

# 1. What is MapReduce

MapReduce is a programming model for data processing. It is a software framework designed for processing huge volumes of data in parallel by dividing the task into a set of independent tasks that will run on distributed commodity hardware.

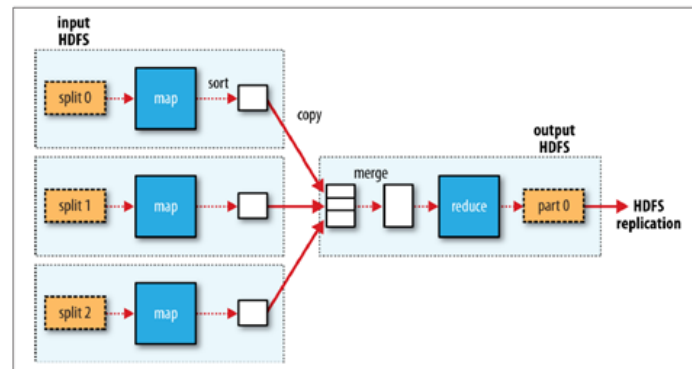


Figure 3.14: MapReduce data flow with a single reduce task (source: "Hadoop The Definitive Guide")

MapReduce works by breaking the processing into two phases: the **map** phase and the **reduce** phase. **Each phase has key-value pairs as input and output** whose types are chosen by the programmer.

The programmer also specifies two functions: the map function and the reduce function.

## 1. The map function

It maps input key/value pairs to a set of intermediate key/value pairs. More specifically, in the **map** phase the user-defined map function processes the input data. The map phase breaks large input files into multiple chunks (input splits), processing each chunk simultaneously with multiple mapper processes. The resulting output from the different chunks is partitioned into sets, which are sorted on the local disks. The map function is represented by the Mapper class, which declares an abstract **map() method**. The Mapper class is a generic type, with four formal type parameters that specify the input key, input value, output key, and output value types of the map function.

Consider that the output pairs do not need to be of the same types as input pairs. So, a given input pair may map to zero or many output pairs.

The total size of input data, i.e. the total number of blocks of the input files defines the required number of map processes. As a general rule, the correct level of parallelism for maps is around 10-100 map processes per node. However, in the case of very CPU-light tasks 300 maps have been set up.

## 2. The reduce function

It reduces a set of intermediate values which share a key to a smaller set of values. More specifically, the **reduce** phase takes each of the sorted chunks, merges and processes them, creating the final output file. In the Reduce phase, the user-defined reduce function processes the Mappers output and generates the final results.

Between the Map and Reduce phases, exists a phase called **Shuffle and Sort** in MapReduce.

The number of required reducers can be identified as follows:

- $0.95 * \text{<no. of nodes>} * \text{<no. of maximum containers per node>}$ . In this case all reduces [\[1\]](#), [\[2\]](#). can launch immediately and start transferring map outputs as the maps finish.

or

- $1.75 * \text{<no. of nodes>} * \text{<no. of maximum containers per node>}$ . In this case, the faster nodes will finish their first round of reduces [\[3\]](#) and launch a second set of reduces doing a much better job of load balancing.

It is obvious that by increasing the number of reduces, [\[4\]](#). Hadoop's overhead is increased; on the other hand, the load balancing is increased and the cost of failures is lowered.

A **MapReduce Job** or a "full program" is a work that the client wants to be performed. It includes the execution of a Mapper and Reducer across a data set. A MapReduce job consists of the input data, the MapReduce Program, and configuration info.

Hadoop can run MapReduce programs written in various languages, such as Java, Ruby, and Python. MapReduce programs are inherently parallel, thus putting very large-scale data analysis into the hands of anyone with enough machines at their disposal.

A **MapReduce task** is an execution of a Mapper or a Reducer on a slice of data. As it is likely that while processing data any machine can go down, MapReduce reschedules the task to some other node. There is an upper limit for task rescheduling to another node. The default value is 4. So, if a task (Mapper or reducer) fails 4 times, then the job is considered as failed. However, in the case of high priority jobs this limit can be increased.

## 2. MapReduce job execution flow

The MapReduce job execution flow is depicted at the following Figure.

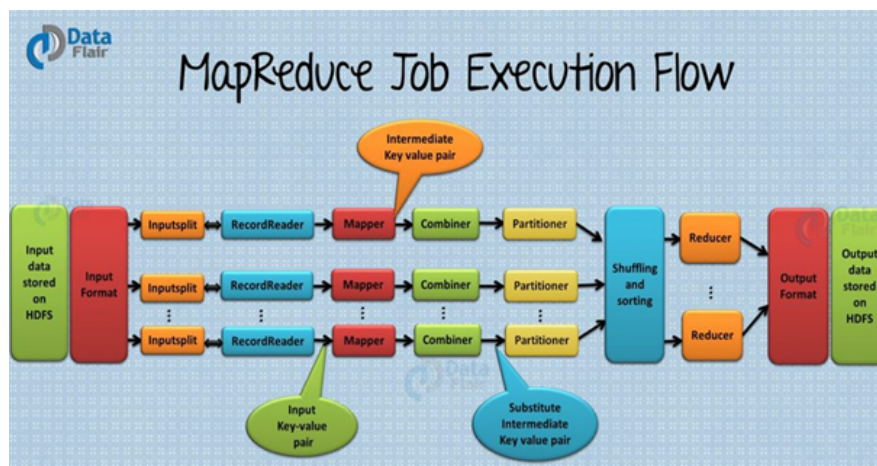


Figure 3.15: MapReduce job execution flow (source: <https://data-flair.training/blogs/how-hadoop-mapreduce-works/>)

Initially, the data for a MapReduce task is stored in input files that reside on HDFS. Their format is arbitrary, while line-based log files and binary format can also be used.

### InputFormat

The first component in MapReduce is the InputFormat. The InputFormat defines how the input files are split and read. It is responsible for creating the InputSplits and dividing them into records, as well as for selecting the files or other objects that are used for input during data processing. The functionalities provided by the InputFormat class are the following:

- Validate the input-specification of the job,
- Split-up the input file(s) into logical InputSplits, each of which is then assigned to an individual Mapper and
- Provide the RecordReader implementation to be used to collect input records from the logical InputSplit for processing by the Mapper.

There are many types of InputFormat in Hadoop such as FileInputFormat, TextInputFormat, KeyValueTextInputFormat, etc.

The default behavior of file-based InputFormat implementations, typically sub-classes of FileInputFormat, is to split the input into logical InputSplit instances based on the total size, in bytes, of the input files. If the TextInputFormat is the InputFormat for a given job, Hadoop detects input-files with the .gz extensions and automatically decompresses them. However, the compressed files with the above extensions cannot be split and each compressed file is processed in its entirety by a single mapper.

### InputSplits

The InputSplit represents the data to be processed by an individual Mapper. Typically, it presents a byte-oriented view on the input. The split is divided into records. Hence, the mapper processes each record which is a key-value pair. The RecordReader of the job is responsible to process the InputSplits and present a record-oriented view. Given that one Mapper is created for each InputSplit the number of mapper tasks is equal to the number of InputSplits. Notice that an InputSplit does not contain the input data and is a reference to the input data.

The length of the InputSplit is measured in bytes. Every InputSplit has storage locations that are used by MapReduce to place the mapper tasks as close to the split's data as possible. In order to minimize the job runtime, the mapper tasks are processed in the order of the size of the splits so that the largest one is processed first.

### RecordReader

The RecordReader communicates with the inputSplit and converts the data into key-value pairs in a format that is suitable for reading by the Mapper. By default, the RecordReader uses the TextInputFormat to convert data into a key-value pair. The "start" is the byte position in the file where the RecordReader should start generating key-value pairs and the "end" is where it should stop reading records.

### Mapper

The Mapper processes the input key-value pairs produced by the RecordReader and generates intermediate key-value pairs. This intermediate output is completely different from the input pair. The Hadoop framework does not store the intermediate key value pairs on HDFS, as this data is temporary and writing on HDFS will create unnecessary multiple copies. The intermediate key-value

pairs are passed by the Mapper to the Combiner for further processing.

### **Combiner**

The Combiner performs local aggregation on the mapper's output aiming to minimize the data transfer between mapper and reducer. So, when the combiner functionality completes, the framework passes the output to the partitioner for further processing.

### **Partitioner**

The Partitioner is used when we are working with more than one reducer. A partitioner partitions the key-value pairs of intermediate Map-outputs. It partitions the data using a user-defined condition, which works like a hash function. The total number of partitions is the same as the number of Reducer tasks for the job.

### **Shuffling and sorting**

During shuffling the input to the reducer is the sorted output of the mappers. In this phase the framework fetches the relevant partition of the output of all the mappers. So, the shuffling phase is necessary for the reducers, otherwise, they would not have any input (or input from every mapper). As shuffling can start even before the map phase has finished, the tasks can be completed in a shorter time.

During sorting the framework groups the reducer inputs by keys (since different mappers may have output the same key). Consider that the shuffle and sort phases occur simultaneously, while the map-outputs are being fetched they are merged.

### **Reducer**

The Reducer reduces a set of intermediate values which share a key to a smaller set of values. Then, the output of the reducer is the final output that is stored at HDFS.

### **Output Format**

OutputFormat defines how the RecordReader writes the output key-value pairs in output files. Thus, the OutputFormat writes the final output of the reducer at HDFS.

MapReduce relies on the OutputFormat of the job to:

- Validate the output-specification of the job; for example, check that the output directory doesn't already exist.
- Provide the RecordWriter implementation used to write the output files of the job. Output files are stored in a FileSystem.

The default OutputFormat is the TextOutputFormat.

### 3. MapReduce data processing example

We present an example of MapReduce data processing using a data set containing temperatures from a weather station. The data set has text format, and the key is the offset of the beginning of the line from the beginning of the file. The data set stores various parameters, such as the air temperature, the dew point temperature, the visibility distance and the atmospheric pressure.

A sample of the data set is depicted below

```
0067011990999991950051507004...9999999N9+00001+9999999999...
```

```
0043011990999991950051512004...9999999N9+00221+9999999999...
```

```
0043011990999991950051518004...9999999N9-00111+9999999999...
```

```
0043012650999991949032412004...0500001N9+01111+9999999999...
```

```
0043012650999991949032418004...0500001N9+00781+9999999999...
```

We want to compute for each year the maximum air temperature. The operation of the map and reduce functions is simple.

#### Operation of the map function

The map function extracts the year and air temperature highlighted with bold (as is depicted below) as these are the parameters we are interested in. Note that the input data are presented to the map function as a key-value pair. For example, for the first line of the input data the key is '0' and the value is '0067011990999991950051507004...9999999N9+00001+9999999999...'. The keys are the line offsets within the file, which we ignore in our map function.

```
(0, 0067011990999991950051507004...9999999N9+00001+9999999999...)
```

```
(106, 0043011990999991950051512004...9999999N9+00221+9999999999...)
```

```
(212, 0043011990999991950051518004...9999999N9-00111+9999999999...)
```

```
(318, 0043012650999991949032412004...0500001N9+01111+9999999999...)
```

```
(424, 0043012650999991949032418004...0500001N9+00781+9999999999...)
```

Thus, the intermediate key-value pairs produced by the map function are the following. Consider that the temperature values have been interpreted as integers.

```
(1950, 0)
```

```
(1950, 22)
```

```
(1950, -11)
```

```
(1949, 111)
```

```
(1949, 78)
```

So, in this example the map function is just a data preparation phase, setting up the data in such a way that the reduce function can do its work on it.

#### The reduce function

The **reduce function** computes the maximum temperature for each year by dropping temperatures that are missing, erroneous or suspect. The output from the map function is processed by the MapReduce framework before being sent to the reduce function. This processing sorts and groups the key-value pairs by key. Thus, the input to the reduce function is the following:

```
(1949, [111, 78])
```

```
(1950, [0, 22, -11])
```

The final output, i.e. the maximum global temperature recorded in each year that is stored at HDFS are the following.

```
(1949, 111)
```

```
(1950, 22)
```

The whole data processing is depicted at Figure 3.16.

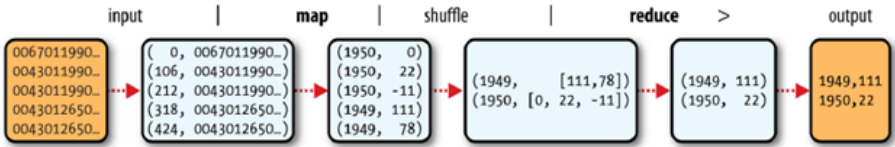


Figure 3.16: MapReduce data processing example (source: Tom White, "Hadoop, The Definitive Guide, Storage and Analysis at Internet scale", 4<sup>th</sup> edition, Oreilly).

To run this example, we need a map function, a reduce function, and some code to run the MapReduce job. The map and reduce functions can be written using various programming languages, like Java, C++ and Python.



## 4. How Hadoop runs a MapReduce job

Figure 3.17 describes how Hadoop runs a MapReduce job. In the process, 5 major entities of Hadoop are involved:

- The client, which submits the MapReduce job.
- The YARN resource manager, which coordinates the allocation of computing resources on the cluster.
- The YARN node managers, which launch and monitor the computing containers on the machines of the cluster.
- The MapReduce application master, which coordinates the tasks running the MapReduce job. The application master and the MapReduce tasks run in containers that are scheduled by the resource manager and managed by the node managers.
- HDFS which is used to store job files (e.g. input file to mapper and output file of reducer) between the other entities.

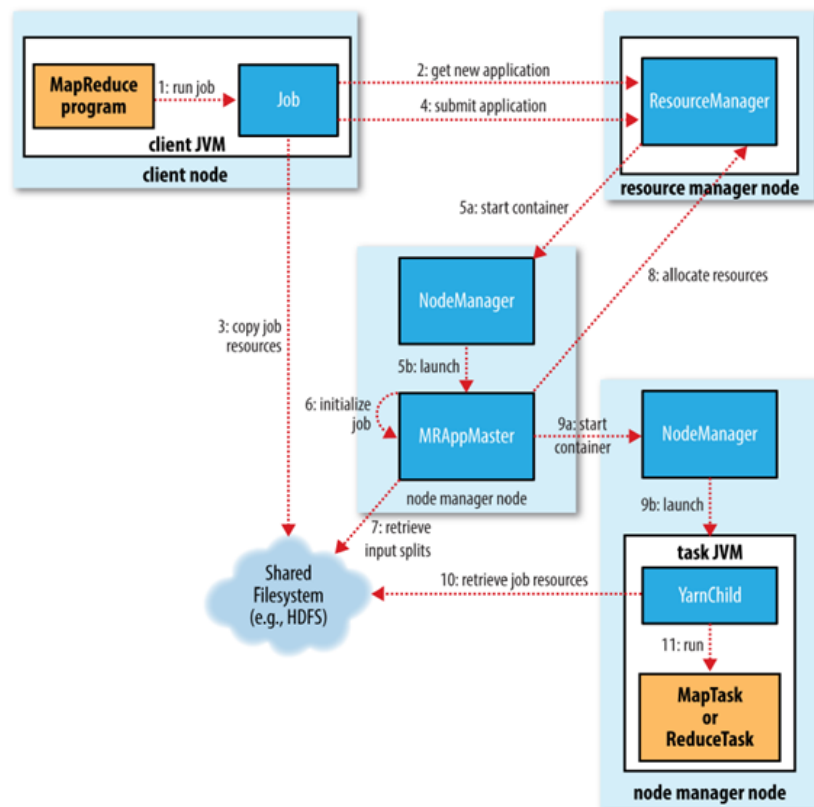


Figure 3.17: How Hadoop works (source: Hadoop with Python, O'Reilly Media, Inc)

The process can be summarized as follows:

- HDFS divides the client input data into 128 MB blocks. Copies of blocks are created based on the replication factor. The blocks and their replicas are stored on different HDFS DataNodes.
- Once all blocks are stored on HDFS DataNodes, the user can process the data.
- The client submits the MapReduce program to Hadoop (specifically to the Resource Manager) to process the data.
- The Resource Manager allocated the MapReduce program on individual nodes in the cluster.
- Once all nodes complete processing, the output is written back to the HDFS.

## 5. Hadoop streaming

Hadoop streaming is a tool that is part of the Hadoop distribution. It allows creating MapReduce jobs with any executable as the mapper and/or the reducer. The Hadoop streaming utility enables Python, shell scripts, or any other language to be used as a mapper, reducer, or both.

Both the mapper and the reducer are executables that read the input from stdin (line by line) and write the output to stdout. The streaming tool will create a Map/Reduce job, submit the job to an appropriate cluster, and monitor the progress of the job until it completes.

When an executable is specified for mappers, each mapper task will launch the executable as a separate process when the mapper is initialized. As the mapper task runs, it converts its inputs into lines and feeds the lines to the stdin of the process. In the meantime, the mapper collects the line-oriented outputs from the stdout of the process and converts each line into a key/value pair, which is collected as the output of the mapper. By default, the prefix of a line up to the first tab character is the key and the rest of the line (excluding the tab character) will be the value. If there is no tab character in the line, then the entire line is considered as key, and the value is null (note that this can be customized).

When an executable is specified for reducers, each reducer task will launch the executable as a separate process once the reducer is initialized. As the reducer task runs, it converts its input key/value pairs into lines and feeds the lines to the stdin of the process. In parallel, the reducer collects the line-oriented outputs from the stdout of the process, converts each line into a key/value pair, which is collected as the output of the reducer. By default, the prefix of a line up to the first tab character is the key and the rest of the line (excluding the tab character) is the value (this can also be customized).

So Hadoop streaming is a utility that allows you to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer.

To run a Map/Reduce job with any executable or script (e.g. Python) as the mapper/reducer use the following command:

```
$HADOOP_HOME/bin/hadoop jar $HADOOP_HOME/hadoop-streaming.jar \
```

```
-input HDFS_input_Dirs \
-output HDFS_output_Dir \
-mapper mapper_executable \
-reducer reducer_executable
```

where:

- HDFS\_input\_Dirs: HDFS directory with the input for the mapper
- HDFS\_output\_Dir: HDFS directory with the output of the reducer
- mapper\_executable: The executable of the mapper
- reducer\_executable: The executable of the reducer

Please visit this [site](#) to see other available streaming options

## 6. Running a simple MapReduce job

We will use the hadoop streaming utility to run a simple MapReduce job. First, we should find the location of the hadoop streaming tool in hadoop installation.

```
$ cd hadoop/hadoop-3.3.1/
```

```
$ find -name hadoop-streaming-*.jar
```

```
./share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar
```

```
./share/hadoop/tools/sources/hadoop-streaming-3.3.1-sources.jar
```

```
./share/hadoop/tools/sources/hadoop-streaming-3.3.1-test-sources.jar
```

So in the above case the Hadoop streaming utility (i.e the file 'hadoop-streaming-3.3.1.jar') is under the path /share/hadoop/tools/lib.

To run a simple MapReduce job we will use as input a sample [text file](#).

1. Download the aforementioned text file at your local machine.

```
$ wget https://www.gutenberg.org/cache/epub/20417/pg20417.txt
```

2. Copy the file under the /user HDFS directory. In the case that the directory does not exist it should be created.

```
$ hdfs dfs -put pg20417.txt /user
```

3. Run the MapReduce job using the Hadoop streaming utility

```
$ hadoop jar hadoop/hadoop-3.3.1/share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar \
```

```
-input /user/pg20417.txt \
```

```
-output /user/output \
```

```
-mapper /bin/cat \
```

```
-reducer /usr/bin/wc
```

The input to the map phase is the txt file pg20417.txt stored at HDFS, while the output of the reduce phase, thus the output of the execution of the MapReduce job is stored at the folder /user/output.

Take into account that in this example, there is no need to write the mapper and reducer programs, as the job uses as a mapper the cat command and as a reducer the wc command, which both are built in any Linux distribution. So, what the mapper does is to echo the input, while what the reducer does is to count the number of lines, words and characters of the text file.

Below is a sample of the messages depicted after job submission.

```
2021-11-11 06:06:25,118 INFO client.DefaultNoHARMFaiOverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
```

```
2021-11-11 06:06:25,358 INFO client.DefaultNoHARMFaiOverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032
```

```
2021-11-11 06:06:25,930 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/vgkamas/.staging/job_1636612827779_0007
```

```
2021-11-11 06:06:26,853 INFO mapred.FileInputFormat: Total input files to process : 1
```

```
2021-11-11 06:06:27,043 INFO mapreduce.JobSubmitter: number of splits:2
```

```
2021-11-11 06:06:27,655 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1636612827779_0007
```

```
2021-11-11 06:06:27,655 INFO mapreduce.JobSubmitter: Executing with tokens: []
```

```
2021-11-11 06:06:28,084 INFO conf.Configuration: resource-types.xml not found
```

```
2021-11-11 06:06:28,084 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
```

```
2021-11-11 06:06:28,653 INFO impl.YarnClientImpl: Submitted application application_1636612827779_0007
```

2021-11-11 06:06:28,967 INFO mapreduce.Job: The url to track the job: http://ubuntu:8088/proxy/application\_1636612827779\_0007/

2021-11-11 06:06:28,975 INFO mapreduce.Job: Running job: job\_1636612827779\_0007

2021-11-11 06:06:47,703 INFO mapreduce.Job: Job job\_1636612827779\_0007 running in uber mode : false

2021-11-11 06:06:47,713 INFO mapreduce.Job: map 0% reduce 0%

2021-11-11 06:07:22,828 INFO mapreduce.Job: map 50% reduce 0%

2021-11-11 06:07:23,835 INFO mapreduce.Job: map 100% reduce 0%

2021-11-11 06:07:31,907 INFO mapreduce.Job: map 100% reduce 100%

2021-11-11 06:07:32,928 INFO mapreduce.Job: Job job\_1636612827779\_0007 completed successfully

From the above messages it is depicted that the input file is broken into 2 input splits, while the ID assigned to the job is 'job\_1636612827779\_0007'. As the last message says, the job has been completed successfully. At the HDFS folder /user/output you will see 2 files, an empty file called \_SUCCESS and the file part-\* that contains the output of the job.

**\$ hdfs dfs -ls /user/output**

**Found 2 items**

```
-rw-r--r-- 1 vgakamas supergroup 0 2021-11-11 06:07 /user/output/_SUCCESS
-rw-r--r-- 1 vgakamas supergroup 25 2021-11-11 06:07 /user/output/part-00000
```

To display the contents of the file part-00000 run the following command.

**\$ hdfs dfs -cat /user/output/part-00000**

```
12760 109767 674567
```

To download the file part-00000 locally from HDFS run the following command

**\$ hdfs dfs -get /user/output/part-00000**

To run the same MapReduce job using 4 mappers use the following command.

```
$ hadoop jar hadoop/hadoop-3.3.1/share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar \
-D mapreduce.job.maps=4 \
-input /user/pg20417.txt \
-output /user/output \
-mapper /bin/cat \
-reducer /usr/bin/wc
```

Take into account that you should delete the /user/output HDFS folder

**\$ hdfs dfs -rm -R /user/output**

or define a new HDFS folder to store the outputs of the execution of MapReduce jobs.

# Hadoop Administration

Site: [DTAM Online Training Platform](#)  
Course: Big Data  
Book: Hadoop Administration

Printed by: Vasileios Gkamas  
Date: Friday, 27 October 2023, 4:22 PM

## Table of contents

1. Main daemons and configuration files
2. Common commands for Hadoop administration

# 1. Main daemons and configuration files

The main daemons running at a Hadoop cluster are the following:

- NameNode
- DataNode
- Secondary NameNode
- ResourceManager
- NodeManager

The main Hadoop configuration files are the following:

File	Description
hadoop-env.sh	Sets ENV variables for Hadoop
core-site.xml	Parameters for entire Hadoop cluster
hdfs-site.xml	Parameters for HDFS and its clients
mapred-site.xml	Parameters for MapReduce and its clients
yarn-site.xml:	Parameters for YARN

The main directories of Hadoop installation are the following:

- **etc/hadoop/**: It contains the Hadoop configuration files
- **bin**: It contains the executables of Hadoop, i.e. yarn, hdfs and mapred
- **sbin**: It contains script for running/stopping Hadoop services

## 2. Common commands for Hadoop administration

Below are presented common commands for Hadoop administration.

Note that with these commands the `$HADOOP_HOME` should be replaced with the home path of Hadoop installation.

An alternative option is to edit the file `.bashrc` (e.g. running the command "`vi ~/.bashrc`") located at your home directory and add the following lines.

- `export JAVA_HOME=/usr/lib/jvm/java-1.8.0-openjdk-i386`
- `export HADOOP_HOME=~/.hadoop/hadoop-3.3.1`
- `export PATH=$PATH:$HADOOP_HOME/bin`
- `export HADOOP_CONF_DIR=$HADOOP_HOME/etc/Hadoop`

Be sure that you define the correct paths, e.g. the path to the HADOOP home directory.

Then run the following command to source the latest variables:

```
$ source ~/.bashrc
```

1. Check the running Hadoop daemons

```
$ jps
```

2. Start the YARN daemon.

```
$ $HADOOP_HOME/sbin/start-yarn.sh
```

3. Start the HDFS daemon

```
$ $HADOOP_HOME/sbin/start-dfs.sh
```

4. Stop the YARN daemon

```
$ $HADOOP_HOME/sbin/stop-yarn.sh
```

5. Stop the HDFS daemon

```
$ $HADOOP_HOME/sbin/stop-dfs.sh
```

6. Check the version of Hadoop

```
$ $HADOOP_HOME/bin/hadoop version
```

7. Check the health of the Hadoop file system

```
$ $HADOOP_HOME/bin/hdfs fsck /
```

8. Display files during checking the health of Hadoop file system

```
$ $HADOOP_HOME/bin/hdfs fsck / -files
```

9. Delete corrupted files from Hadoop file system

```
$ $HADOOP_HOME/bin/hdfs fsck -delete
```

10. Format the NameNode. Note that the first time you bring up HDFS, it must be formatted. Format a new distributed file system as hdfs:

```
$ $HADOOP_HOME/bin/hdfs namenode -format
```

11. Turn off the safe mode of NameNode.

```
$ $HADOOP_HOME/bin/hdfs dfsadmin -safemode leave
```



# Hadoop MapReduce with Python

Site: [DTAM Online Training Platform](#)  
Course: Big Data  
Book: Hadoop MapReduce with Python

Printed by: Vasileios Gkamas  
Date: Friday, 27 October 2023, 4:22 PM

## Table of contents

1. Writing a MapReduce Program in Python using the Hadoop streaming utility
2. Writing a MapReduce Program in Python using mrjob

# 1. Writing a MapReduce Program in Python using the Hadoop streaming utility

To demonstrate the Hadoop streaming utility, we will write a Python application that counts the instances of every word appearing in a text file. We will implement this application as two different Python programs, a mapper and a reducer.

The **mapper.py** is the Python program that implements the mapper phase of the Python application. The mapper reads data from standard input (stdin), splits the lines into words, and outputs each word with its intermediate count to the standard output (stdout). The code of the mapper.py script is shown below.

```
#!/usr/bin/env python

"""mapper.py"""

import sys

input comes from STDIN (standard input)

for line in sys.stdin:

 # remove leading and trailing whitespace. The Python strip() method removes any
 # spaces or specified characters at the start and end of a string

 line = line.strip()

 # split the line into words. The Python split() method splits a string into a list where
 # each word is a list item

 words = line.split()

 # increase counters

 for word in words:

 # write the results to STDOUT. This will be the input for the Reduce step

 # tab-delimited; the trivial word count is 1

 print '%s\t%s' % (word, 1)
```

The **reducer.py** is the Python program that implements the reducer phase of the Python application. The reducer reads the results of mapper.py from the standard input (stdin), sums the occurrences of each word, and writes the result to the standard output (stdout). The code of the reducer.py script is shown below.

```
#!/usr/bin/env python

"""reducer.py"""

from operator import itemgetter

import sys

current_word = None

current_count = 0

word = None

input comes from STDIN

for line in sys.stdin:

 # remove leading and trailing whitespace

 line = line.strip()

 # parse the input we got from mapper.py
```

```

word, count = line.split('\t', 1)

convert count (currently a string) to int

try:

 count = int(count)

except ValueError:

 # count was not a number, so silently
 # ignore/discard this line

 continue

this IF-switch only works because Hadoop sorts map output
by key (here: word) before it is passed to the reducer

if current_word == word:

 current_count += count

else:

 if current_word:

 # write result to STDOUT

 print '%s\t%s' % (current_word, current_count)

 current_count = count

 current_word = word

do not forget to output the last word if needed!

if current_word == word:

 print '%s\t%s' % (current_word, current_count)

```

The overall MapReduce process is depicted at the following figure.

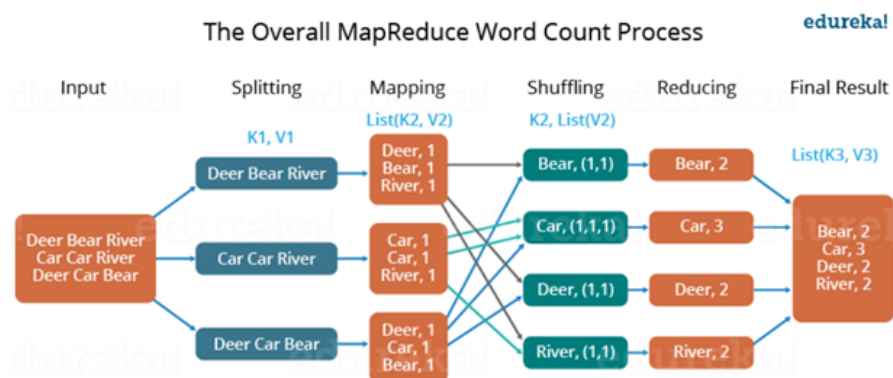


Figure 3.18: Example of MapReduce process for the WordCount example (source: <https://www.edureka.co/blog/mapreduce-tutorial/>)

Before running the Python application at the Hadoop, we check the execution of the mapper and reducer locally. It is important to grant the execute permission at the mapper and reducer:

```
$ chmod +x mapper.py
```

```
$ chmod +x reducer.py
```

Verify execution of mapper.py:

```
$ echo 'hello everybody day hello day hello' | ./mapper.py
```

```
hello 1
```

```
everybody 1
```

```
day 1
```

```
hello 1
```

```
day 1
```

```
hello 1
```

Verify execution of reducer.py:

```
$ echo 'hello everybody day hello day hello' | ./mapper.py | sort -t | ./reducer.py
```

```
day 2
```

```
everybody 1
```

```
hello 3
```

Once you have verified the correct execution of the mapper and reducer locally, we can run the MapReduce program. Initially we copy under our home directory at HDFS (i.e. /user/hdpuser) the input file for the mapper. In our case it is named testfile.txt and contains 4 lines with text.

```
$ cat testfile.txt
```

```
hello world everybody day world
```

```
world hello good morning day world
```

```
everybody world day hello everyday
```

```
everybody everyday world
```

```
$ hdfs dfs -put testfile.txt /user/hdpuser
```

Then we run the MapReduce program using the following command

```
hadoop jar hadoop/hadoop-3.3.1/share/hadoop/tools/lib/hadoop-streaming-3.3.1.jar \
```

```
-file /home/vgkamas/mapper.py -mapper /home/vgkamas/mapper.py \
```

```
-file /home/vgkamas/reducer.py -reducer /home/vgkamas/reducer.py \
```

```
-input /user/hdpuser/testfile.txt -output /user/hdpuser/wordcount
```

At this command take into account that you may have to replace the path of the Hadoop streaming jar with the one that is applicable in your Hadoop installation. Then you can copy from HDFS at your home directory at the local system the output of the reducer and verify the contents of the output.

```
$ hdfs dfs -get /user/hdpuser/wordcount /home/vgkamas/wordcount
```

After the successful execution of the MapReduce job you can go to the User Interface of YARN running at <http://<yarn-server>:8088/cluster> to see the completed job and see more information. For example:

The screenshot shows the Hadoop YARN web interface. The top navigation bar includes the Hadoop logo and the application ID: **Application application\_1636358143346\_0004**. The left sidebar contains a navigation menu with options like Cluster, Nodes, Node Labels, Applications, and Tools. The main content area displays the application overview, including details such as User (vgkamas), Name (streamjob7827804672235305714.jar), Application Type (MAPREDUCE), Application Priority (0), and YarnApplicationState (FINISHED). It also shows the FinalStatus Reported by AM (SUCCEEDED), Started time (Mon Nov 08 02:26:30 -0800 2021), Launched time (Mon Nov 08 02:26:35 -0800 2021), and Finished time (Mon Nov 08 02:27:40 -0800 2021). The bottom section displays Application Metrics, including Total Resource Preempted, Total Number of Non-AM Containers Preempted, and Aggregate Resource Allocation.

Figure 3.19: Job execution example

## 2. Writing a MapReduce Program in Python using mrjob

mrjob is a python library for writing MapReduce code using the Python programming language. Using mrjob, the developers can test the MapReduce Python code locally on their system, on a Hadoop cluster and on the cloud using Amazon Elastic MapReduce. Mrjob is extensively integrated with Amazon Elastic MapReduce, so it is as easy to run your job in the cloud as it is to run it on your laptop.

mrjob is installed using pip by running the following command.

**\$ pip install mrjob**

If you have installed Python 3.x then run the following command

**\$ pip3 install mrjob**

Keep all MapReduce code for one job in a single class.

Open a new file and write the following Python code. This Python code computes the number of characters, words and lines of a text file.

```
from mrjob.job import MRJob
```

```
class MRWordFrequencyCount(MRJob):
```

```
 def mapper(self, _, line):
```

```
 yield "chars", len(line)
```

```
 yield "words", len(line.split())
```

```
 yield "lines", 1
```

```
 def reducer(self, key, values):
```

```
 yield key, sum(values)
```

```
if __name__ == '__main__':
```

```
 MRWordFrequencyCount.run()
```

The MapReduce job is defined as the class, MRWordFrequencyCount. Within the mrjob library, the class that inherits from MRJob contains the methods that define the steps of the MapReduce job.

The mapper() function defines the mapper for MapReduce and takes as input a key-value argument and generates the output in tuple format (output\_key, output\_value) . The mapper in this example ignores the input key. It is splitting the line and generating a word with its own count.

The reducer() function is aggregating the result according to their key and producing the output in a key-value format with its total count.

The 2 lines at the end of the script are ensuring the execution of mrjob. These lines pass control over the command line arguments and execution to mrjob. Without them, your job will not work.

To run the mrjob locally execute the following command, where:

- script\_name is the name of the Python script you have written and
- input.txt is the input file.

**\$ python script\_name.py input.txt**

By default, mrjob runs locally, allowing to write and debug code before submitted to a Hadoop cluster.

The -r option defines how the job is running. The different values that this option takes are the following:

- -r inline: This is the default option. Run in a single Python process
- -r local: Run locally in a few subprocesses simulating some Hadoop features
- -r hadoop: Run on a Hadoop cluster
- -r emr: Run on Amazon Elastic Map Reduce

For example, to run a Python code named `estimval.py` at a Hadoop cluster run the following command.

```
$ python estimval.py -r hadoop hdfs:///user/hadoop/input.txt
```

Consider that you have first to upload at the defined HDFS directory (i.e. `/user/hadoop` in the above example) the input file that the Python script uses. This is done by using the `"hdfs dfs -put"` command.



# Hadoop and Pig

Site: [DTAM Online Training Platform](#)  
Course: Big Data  
Book: Hadoop and Pig

Printed by: Vasileios Gkamas  
Date: Friday, 27 October 2023, 4:22 PM

## Table of contents

1. What is Pig
2. Pig installation
3. Pig Latin
4. Writing a Pig script
5. Running a Pig script

# 1. What is Pig

Pig is a high-level programming language useful for analyzing big data sets. In the MapReduce framework, the programs need to be translated into a series of Map and Reduce stages using well-known programming languages such as Java and Python. However, the data analysts are not familiar with this programming model. To cover this gap, an abstraction called Pig was built on top of Hadoop.

Apache Pig enables data analysts to focus more on analyzing big data sets rather than spending time on writing MapReduce programs. Compared to MapReduce programs written in Java, Pig is easier to write, understand and maintain given that it is a data transformation language allowing to describe the processing of data as a sequence of transformations. Pig provides User Defined Functions that allow custom processing to be written in various programming languages, like Python.

Pig consists of two components:

- Pig Latin, which is a high-level data flow language which uses familiar keywords from data processing e.g., Join, Group and Filter
- A runtime environment that parses, optimizes, and executes the Pig Latin scripts as a series of MapReduce jobs that are run on a Hadoop cluster.

A Pig Latin program consists of a series of transformations or operations that are applied for data processing. These operations describe a data flow that is translated by the Hadoop Pig runtime environment into an executable representation. The results of these transformations are a series of MapReduce jobs which a programmer is unaware of.

## 2. Pig installation

For Pig installation follow the steps below.

1. Download the latest Pig binary from <https://dlcdn.apache.org/pig/>

**\$ wget <https://dlcdn.apache.org/pig/pig-0.17.0/pig-0.17.0.tar.gz>**

2. Create a directory with the name Pig at the hadoop installation directory

**\$ mkdir Pig**

3. Copy the pig tar.gz file at the Pig directory. For example

**\$ cp pig-0.17.0.tar.gz hadoop/hadoop-3.3.1/Pig**

Assuming that the Hadoop directory is ~hadoop/ hadoop-3.3.1.

4. Untar the tar.gz file in the Pig directory and move the contents of the directory pig-0.17.0 to the Pig directory

**\$ tar -xvf pig-0.17.0.tar.gz**

5. Go to your home directory and add the following lines at the .bashrc file.

**export PIG\_HOME=~/.hadoop/hadoop-3.3.1/Pig**

**export PATH=\$PATH:~/.hadoop/hadoop-3.3.1/Pig/bin**

**export PIG\_CLASSPATH=HADOOP\_HOME/conf**

More specific we define the following variables:

- PIG\_HOME: directory to the installation folder of Apache Pig
- PATH: environment variable to the bin folder of Apache Pig
- PIG\_CLASSPATH: environment variable to the conf folder of Hadoop installation. It is supposed that you have already defined at the same file the value of HADOOP\_HOME environment variable.

6. Source the file

**\$ source .bashrc**

Check that Pig has been installed correctly

**\$ pig -version**

**Apache Pig version 0.17.0 (r1797386)**

**compiled Jun 02 2017, 15:41:58**

### 3. Pig Latin

Pig Latin is a high-level data flow language, allowing those new to the language to understand and write basic Pig scripts.

The basic constructs that are used to process data in Pig are the statements. Each statement is an operator which takes a relation as an input, performs a transformation on that relation, and produces a relation as an output. Statements can span multiple lines, but all statements must end with a semicolon (;).

A Pig script has the following general structure:

- A LOAD statement that reads the data from HDFS
- One or more statements for data transformation
- A STORE or DUMP statement to store or view the results at HDFS, respectively

Except LOAD and STORE statements, while performing all other operations, Pig Latin statements take a relation as input and produce another relation as output, where a relation is the outermost structure of the Pig Latin data model.

The following table describes the relational operators of Pig Latin.

Operator	Description
<b>Loading and Storing</b>	
LOAD	Load the data from the file system (local/HDFS) into a relation.
STORE	Save a relation to the file system (local/HDFS).
<b>Filtering</b>	
FILTER	Remove unwanted rows from a relation.
DISTINCT	Remove duplicate rows from a relation.
FOREACH, GENERATE	Generate data transformations based on columns of data.
STREAM	Transform a relation using an external program.
<b>Grouping and Joining</b>	
JOIN	Join two or more relations.
COGROUP	Group the data in two or more relations.
GROUP	Group the data in a single relation.
CROSS	Create the cross product of two or more relations.
<b>Sorting</b>	
ORDER	Arrange a relation in a sorted order based on one or more fields (ascending or descending).
LIMIT	Get a limited number of tuples from a relation.
<b>Combining and Splitting</b>	
UNION	Combine two or more relations into a single relation.
SPLIT	Split a single relation into two or more relations.
<b>Diagnostic Operators</b>	
DUMP	Print the contents of a relation on the console.
DESCRIBE	Describe the schema of a relation.

EXPLAIN	View the logical, physical, or MapReduce execution plans to compute a relation.
ILLUSTRATE	View the step-by-step execution of a series of statements.

The syntax of some common operators is the following.

1. **LOAD** operator

LOAD 'data' [USING function] [AS schema];

2. **JOIN** operator

JOIN relation1 BY columnname, relation2 BY columnname;

3. **FILTER** operator

Relation2\_name = FILTER Relation1\_name BY (condition);

4. **FOREACH** operator

Relation\_name2 = FOREACH Relatin\_name1 GENERATE (required data);

Please refer to <https://pig.apache.org/> for more information about the syntax of the other operators

## 4. Writing a Pig script

Below is a simple Pig script that implements the word count example depicted in a previous section.

```
%default INPUT '/user/testfile.txt';

%default OUTPUT '/user/output';

records = LOAD '$INPUT';

terms = FOREACH records GENERATE FLATTEN(TOKENIZE((chararray) $0)) AS word;

grouped_terms = GROUP terms BY word;

word_counts = FOREACH grouped_terms GENERATE COUNT(terms), group;

STORE word_counts INTO '$OUTPUT';
```

This following table provides a description of the statements used at the above Pig script.

Statement	Description
records = LOAD '\$INPUT';	This statement loads data from the filesystem and stores it in the relation records.
terms = FOREACH records GENERATE FLATTEN(TOKENIZE((chararray) \$0)) AS word;	This statement splits each line of text using the TOKENIZE function and eliminates nesting using the FLATTEN operator.
grouped_terms = GROUP terms BY word;	This statement uses the GROUP operator to group the tuples that have the same field.
word_counts = FOREACH grouped_terms GENERATE COUNT(terms), group;	This statement iterates over all of the terms in each bag (i.e. a collection of tuples) and uses the COUNT function to return the sum.
STORE word_counts INTO '\$OUTPUT';	This statement stores the results in HDFS.

## 5. Running a Pig script

You can run Apache Pig in two modes:

- **Local Mode.** In this mode, all files are installed and run from your localhost and local file system. This mode is generally used for testing purposes.
- **MapReduce Mode.** In this mode, we load or process the data that exists in HDFS using Apache Pig. Whenever we execute the Pig Latin statements to process the data, a MapReduce job is invoked in the backend to perform a particular operation on the data stored at HDFS.

To run Apache pig at local mode, use the following command

```
$ pig -x local pigfile.pig
```

While, to run Apache pig at MapReduce mode, use any of the 2 commands below.

```
$ pig - pigfile.pig
```

or

```
$ pig -x mapreduce pigfile.pig
```

Apache Pig scripts can be executed in three ways, namely, interactive mode, batch mode, and embedded mode.

- **Interactive Mode.** At this mode Apache Pig runs by using the Grunt shell. In this shell, you can enter the Pig Latin statements and get the output. Once the Grunt shell is initialized, Pig Latin statements can be entered and executed in an interactive manner
- **Batch Mode.** At this mode Apache Pig runs by writing the Pig Latin script in a single file with .pig extension (like we did in the 2 examples above).
- **Embedded Mode.** At this mode, Apache Pig supports the definition of User Defined Functions in programming languages such as Java and using them in our script.



## Exercise: Apache Pig hands-on

Site: [DTAM Online Training Platform](#)  
Course: Big Data  
Book: Exercise: Apache Pig hands-on

Printed by: Vasileios Gkamas  
Date: Friday, 27 October 2023, 4:22 PM

## Table of contents

1. General description
2. Desired objectives
3. Required material
4. Solution

# 1. General description

This exercise aims to make you familiar with writing Apache Pig statements using the interactive mode (grunt shell).

## 2. Desired objectives

The desired objectives of the exercise are the following:

- To become familiar with the Pig tool
- To perform some data processing tasks using the grunt shell of Apache Pig and running Apache Pig at a local machine

### 3. Required material

There is no required material for doing this exercise. What is needed is access at a Hadoop cluster with the Pig tool installed

## 4. Solution

Let's suppose that we have 2 text files one containing the Identification number and manufacturers of machines installed at a factory and another file containing the identification number and number of failures of each machine.

First create the 2 files with the following contents

```
$ cat machines1.txt
```

```
ID,Manufacturer
```

```
1,Bosch
```

```
2,Phillips
```

```
3,Siemens
```

```
4,Siemens
```

```
5,Cisco
```

```
6,Huawei
```

```
7,Cisco
```

```
$ cat machines2.txt
```

```
ID,Failures
```

```
1,5
```

```
2,6
```

```
3,2
```

```
4,3
```

```
5,1
```

```
6,0
```

```
7,2
```

Perform the following data processing tasks using the grunt shell of Apache Pig and running Apache Pig at the local machine.

### 1. Using the JOIN operator join the 2 files based on the identification number of the machines.

```
$ pig -x local
```

```
grunt> file1 = LOAD '/home/vgkamas/pigexercise/machines1.txt' using PigStorage(',') AS (id:int,manufacturer:chararray);
```

```
grunt> file2 = LOAD '/home/vgkamas/pigexercise/machines2.txt' using PigStorage(',') AS (id:int,failures:int);
```

```
grunt> joined = JOIN file1 by (id), file2 by (id);
```

```
grunt>STORE joined INTO '/home/vgkamas/pigexercise/output1';
```

```
grunt>DUMP joined
```

```
(1,Bosch,1,5)
```

```
(2,Phillips,2,6)
```

```
(3,Siemens,3,2)
```

```
(4,Siemens,4,3)
```

```
(5,Cisco,5,1)
```

```
(6,Huawei,6,0)
```

(7,Cisco,7,2)

## 2. Using the FILTER operator, select only the machines with less than 5 failures

```
$ pig -x local
```

```
grunt> file2 = LOAD '/home/vgkamas/pigexercise/machines2.txt' using PigStorage(',') AS (id:int,failures:int);
```

```
grunt> filtered= FILTER file2 BY failures<5;
```

```
grunt> STORE filtered INTO '/home/vgkamas/pigexercise/output2';
```

```
grunt> DUMP filtered;
```

```
(3,2)
```

```
(4,3)
```

```
(5,1)
```

```
(6,0)
```

```
(7,2)
```

3. **Download from this [URL](#) a predictive maintenance dataset.** This dataset contains data from a variety of IoT sensors for predictive maintenance in elevator industry. The dataset contains operation data, in the form of timeseries sampled at 4Hz in high-peak and evening elevator usage in a building. For an elevator car door system we consider: Electromechanical sensors (Door Ball Bearing Sensor), Ambiance (Humidity) and Physics (Vibration).

Below is a sample of the dataset.

```
$ cat predictive-maintenance-dataset.csv
```

```
ball-bearing;humidity;vibration
```

```
93.744;73.999;18
```

```
93.74;73.999;18.001
```

```
93.736;73.998;18.002
```

```
93.732;73.998;18.003
```

```
93.729;73.998;18.004
```

```
93.725;73.997;18.005
```

```
93.721;73.997;18.006
```

```
.....
```

### A) Using the FOREACH operator get the humidity and vibration from all IoT sensors.

```
$ pig -x local
```

```
grunt> maint = LOAD '/home/vgkamas/predictive-maintenance-dataset.csv' USING PigStorage(';') as (ballb:double, hum: double, vibr: double);
```

```
grunt> select_data = FOREACH maint GENERATE hum, vibr;
```

```
grunt> STORE select_data INTO '/home/vgkamas/pigexercise/output3a';
```

```
grunt> DUMP select_data
```

```
.....
```

```
73.999 18.0
```

```
73.999 18.001
73.998 18.002
73.998 18.003
73.998 18.004
73.997 18.005
73.997 18.006
73.997 18.007
73.996 18.008
73.996 18.009
```

.....

Take into account that at the LOAD statement the PigStorage takes as argument the semicolon (;) as the delimiter of the dataset isthis character.

**B) Using the FILTER operator get the records from the IoT sensors whose vibration is between 18.007 and 20.025.**

```
$ pig -x local
```

```
grunt> maint = LOAD '/home/vgkamas/predictive-maintenance-dataset.csv' USING PigStorage(';') as (ballb:double, hum: double, vibr: double);
```

```
grunt> filtered= FILTER maint BY (vibr > 18.007) AND (vibr < 20.025);
```

```
grunt> STORE filtered INTO '/home/vgkamas/pigexercise/output3b';
```

```
grunt> DUMP filtered;
```

```
93.713 73.996 18.008
93.709 73.996 18.009
93.702 73.995 18.011
93.698 73.995 18.012
93.694 73.995 18.013
93.69 73.994 18.014
93.686 73.994 18.015
93.682 73.994 18.016
93.678 73.993 18.017
93.675 73.993 18.018
93.671 73.993 18.019
93.663 73.992 18.021
93.659 73.992 18.022
93.655 73.992 18.023
93.651 73.991 18.024
93.648 73.991 18.025
```

.....



**C) Using the FILTER operator group the records from the IoT sensors by humidity.**

```
$ pig -x local
```

```
grunt> maint = LOAD '/home/vgkamas/predictive-maintenance-dataset.csv' USING PigStorage(';') as (ballb:double, hum: double, vibr: double);
```

```
grunt> grouped= GROUP maint by hum;
```

```
grunt> STORE grouped INTO '/home/vgkamas/pigexercise/output3c';
```

```
grunt>DUMP grouped;
```

```
72.399 {(74.443,72.399,21.003),(74.448,72.399,21.0),(74.449,72.399,21.0),(74.45,72.399,21.0),(74.451,72.399,21.001),
(74.452,72.399,21.001),(74.453,72.399,21.001),(74.454,72.399,21.002),(74.445,72.399,21.001)}
```

```
72.4 {(74.465,72.4,21.005),(74.464,72.4,21.004),(74.463,72.4,21.004),(74.462,72.4,21.004),(74.461,72.4,21.003),(74.461,72.4,21.003),
(74.46,72.4,21.003),(74.455,72.4,21.002),(74.456,72.4,21.002),(74.457,72.4,21.002),(74.458,72.4,21.003),(74.459,72.4,21.003),
(74.432,72.4,21.008),(74.435,72.4,21.006),(74.437,72.4,21.005),(74.44,72.4,21.004),(74.466,72.4,21.005)}
```

```
72.401 {(74.424,72.401,21.011),(74.421,72.401,21.012),(74.418,72.401,21.014),(74.479,72.401,21.008),(74.478,72.401,21.008),(74.47
7,72.401,21.008),(74.476,72.401,21.007),(74.475,72.401,21.007),(74.474,72.401,21.007),(74.473,72.401,21.006),(74.472,72.401,21.006
```

**D) Using the SPLIT operator**, split the initial predictive maintenance dataset into 2 different datasets, one containing the records from IoT sensors with humidity bigger than 20.0 and a second containing the records from IoT sensors with humidity less or equal to 20.0.

```
$ pig -x local
```

```
grunt> maint = LOAD '/home/vgkamas/predictive-maintenance-dataset.csv' USING PigStorage(';') as (ballb:double, hum: double, vibr: double);
```

```
grunt> SPLIT maint into dataset1 if vibr>20.0, dataset2 if vibr<=20.0 ;
```

```
grunt> STORE dataset1 INTO '/home/vgkamas/pigexercise/outputd1';
```

```
grunt> STORE dataset2 INTO '/home/vgkamas/pigexercise/outputd2';
```

```
grunt>DUMP dataset1 ;
```

```
grunt>DUMP dataset2 ;
```

Do the same data transformation (i.e. splitting the file into 2 files) by running Pig at batch mode.

Edit first the Pig script.

```
$ cat splitting.pig
```

```
%default INPUT '/home/vgkamas/predictive-maintenance-dataset.csv';
```

```
%default OUTPUT1 '/home/vgkamas/pigexercise/outputd1';
```

```
%default OUTPUT2 '/home/vgkamas/pigexercise/outputd2';
```

```
maint = LOAD '$INPUT' USING PigStorage(';') as (ballb:double, hum: double, vibr: double);
```

```
SPLIT maint into dataset1 if vibr>20.0, dataset2 if vibr<=20.0 ;
```

```
STORE dataset1 INTO '$OUTPUT1';
```

```
STORE dataset2 INTO '$OUTPUT2';
```

And then run at the local machine the Pig script

```
$ pig -x local splitting.pig
```





Co-funded by the  
Erasmus+ Programme  
of the European Union



# DIGITAL TRANSFORMATION IN ADVANCED MANUFACTURING

## DTAM PROJECT WORK N° 1 *Big Data challenge*

# Deliverable factsheet

<b>Project Number:</b>	621496-EPP-1-2020-1-ES-EPPKA2-SSA
<b>Project Acronym:</b>	DTAM
<b>Project Title:</b>	Digital Transformation in Advanced Manufacturing
<b>Work package:</b>	WP3: DTAM Training Course
<b>Title of Deliverable:</b>	DTAM Big Data challenge (1 <sup>st</sup> version)
<b>Editor(s):</b>	Maria Rigou, Vasileios Gkamas

## Delivery Slip

Version	Date	Comments
0.1	29/06/2022	Structure of the deliverable
1.0	26/10/2023	Final edition

# Contents

1	The Big Data challenge	4
1.1	Presentation of the project	4
1.2	Group challenge	5
1.3	Goals to reach with the challenge	5
1.4	Rubric (evaluation form) for groups of students	6
2	Learning Units reference	8
3	Learning Outcomes	9
4	IoT Lab Resources	10
5	The project step by step	11
6	Expected result example	12
	Answers for the data processing with Python	12
	Answers for the data processing with Apache Pig	18

# 1 The Big Data challenge

## 1.1 Presentation of the project

A manufacturing company runs an Industry 4.0 project and has installed at its machines various kind of sensors utilizing the Internet of Things. The sensors that have installed are collecting 3 parameters: pressure, temperature and moisture as very high or very low values of those parameter can affect the effective operation of the machines or even break a machine. For example, a high temperature can cause a fire or even explosion of machine, while an excessive moisture can create mold and damage the machine.

The production line manager asked the data science team to analyze a data set with 1,000 records collected since the installation of the sensors at the machines. This dataset includes the following parameters:

- **Lifetime.** This is a numerical parameter. It concerns the period in weeks the machine is active
- **Broken.** This is a numerical parameter. It declares if the machine was broken or not in the corresponding lifetime.
- **pressureInd.** This is a numerical parameter. It quantifies the flow of liquid through pipes.
- **moistureInd.** This is a numerical parameter. It expresses the relative humidity in the air.
- **temperatureInd.** This is a numerical parameter. It expresses the temperature each machine works.
- **team.** This is a categorical parameter. It indicates the team using the machine
- **provider.** This is a categorical parameter. It indicated the name of the machine manufacturer

More specifically the production line manager has asked you to make the following analysis:

- Import the pandas, numpy and matplotlib libraries
- Load the dataset (maintenance21.csv)
- Display the first and the last 5 lines of the dataset.
- Use the info() function to see columns, data type per column and number of non-null values count
- Display the sum of null values per column
- Calculate the percentage of missing values in each column (hint: write your own expression using isnull(), sum() and len() functions)

- Delete all rows that contain na values
- Confirm that the dataset has now less lines
- Display all basic descriptive statistics for the dataset (count, mean, std, min, max and quartiles)
- Display a statistics summary (like above) but now only for column with numerical datatypes, such as int, float
- Check for the number of unique values in each column
- Gather all numerical columns in a dataframe and name it num\_cols
- Remove column «broken» from num\_cols
- Load the seaborn library and use it along with matplotlib to create for each column a histogram and a boxplot visualization (hint: try using a for loop to cover each column in num\_cols)

As a next step you are asked to move the dataset at the Hadoop cluster and make the following analysis using Apache Pig.

- Select the machines with temperature bigger than 90.
- Select the machine coming from provider 4.
- Get the temperature, pressure and moisture from all machines.
- Get the temperature, pressure and moisture from all machines coming from provider 4.
- Split the initial data set into 2 different datasets, one containing the records from machines with moisture bigger than 20 and another containing the records from machine with temperature less or equal to 100.

## 1.2 Group challenge

To do this challenge, you will need to form a group with 1 other student. You will be instructed by a teacher on how to perform the challenge. When your challenge is finished it will be evaluated by a teacher using the evaluation rubric (see below).

## 1.3 Goals to reach with the challenge

- Import a dataset into Python, load the appropriate libraries, and perform typical EDA (Exploratory Data Analysis) steps
- Manipulate nonnumeric values and create a dataframe with the numerical values
- create for each column a histogram and a boxplot using seaborn and matplotlib libraries
- Transfer a dataset from/to a Hadoop cluster running at containers
- Interact with HDFS
- Using Apache Pig for data analysis at a Hadoop cluster.

## 1.4 Rubric (evaluation form) for groups of students

After completing the challenge, the instructor(s) will evaluate the work of the students. This rubric will be used to evaluate competences and learning outcomes on a 3-points scale of: Needs improvement, As expected, Good.

Criterion	Needs improvement	As expected	Good
Student is able to load the provided dataset and the required libraries into the available Python installation	Fails to locate and load the dataset <i>and/or</i> Fails to load the required libraries	Loads the provided dataset and the required libraries into the available Python installation but requires some help from the instructor	Loads the provided dataset and the required libraries into the available Python installation
Student is able to explore the data in the dataset and calculate summary statistics.	Fails to write the correct code for exploring the dataset and producing summary stats <i>and/or</i> Partially writes the correct Python code	Explores the dataset and produces summary stats of the data	Is able to also explain the stats produced <i>and/or</i> Experiments with additional data exploration functions
Student is able to create a dataframe and manipulate it	The dataframe is not created <i>and/or</i> The dataframe is not created with the correct structure and contents	The student creates the dataframe and fails to notice the difference between null and na values	Student creates the dataframe and manipulates it using additional Python commands from the ones requested <i>and/or</i> Understands the difference between null values and na values and writes the appropriate code
Student is able to create	either one of the visualizations is	histograms and boxplots are not	Histograms and boxplots are



boxplots and histograms	missing or is not correct <i>and/or</i> visualizations also include nonnumeric columns	created with the requested loop command <i>and/or</i> histograms and boxplots include nonnumeric columns as well	generated correctly and using a loop command
Student is able to install, configure and operate VPN connection (product)	VPN connection not working (product) <i>and/or</i> VPN connection works sometimes but unreliable <i>and/or</i> Student is not able to operate without help from teacher	VPN connection open and works reliable but with many timeouts	VPN connection open and works reliable without problems
Student is able to install, configure and operate SSH client connection (product)	SSH connection not working (product) <i>and/or</i> SSH connection works sometimes but unreliable <i>and/or</i> Student is not able to operate without help from teacher	SSH connection open and works reliable but with many timeouts	SSH connection open and works reliable without problems
Student uploads connects to the container running HDFS and uploads data to HDFS	Connection to the container running HDFS is not working	Connection to the container running HDFS works but data uploading fails	Connection to the container running HDFS and data uploading works
Student uses Apache Pig for data analysis	Student is not able to use Apache Pig without help from teacher	Student is able to use Apache Pig but sometimes the data analysis fails	Student is able to use Apache Pig without problems

## 2 Learning Units reference

These Learning units have to be completed before to approach this Project Work

TRAINING MODULE	LEARNING UNITS
<b>BIG DATA (TM2)</b>	<ul style="list-style-type: none"><li>○ Introduction to Big Data</li><li>○ Python for Data Analysis</li><li>○ The Apache Hadoop framework</li></ul>
<b>MACHINE LEARNING (TM3)</b>	<ul style="list-style-type: none"><li>○</li></ul>
<b>INTERNET OF THINGS AND SENSORS (TM4)</b>	<ul style="list-style-type: none"><li>○</li></ul>
<b>CYBERSECURITY (TM5)</b>	<ul style="list-style-type: none"><li>○</li></ul>
<b>TRANSVERSAL SKILLS (TM6)</b>	<ul style="list-style-type: none"><li>○</li></ul>



### 3 Learning Outcomes

These Skills and Knowledge will be improved upon the project work completion.

TRAINING MODULE	SKILLS AND KNOWLEDGE
BIG DATA (TM2)	<b>Skills</b> <ul style="list-style-type: none"><li>• Importing and Exporting Data in Python</li><li>• Dealing with Missing Values in Python</li><li>• Performing data preprocessing using Python</li><li>• Calculating basic statistics with Python</li><li>• Performing data visualizations using Python</li><li>• Using the Hadoop framework for data storage and processing</li><li>• Using the storage (HDFS) and processing (YARN) services of Hadoop</li><li>• Using Pig programming language to interact with Hadoop</li></ul>
	<b>Knowledge</b> <ul style="list-style-type: none"><li>• How to use the Jupyter Notebook to write Python programs</li><li>• Main Python tools for exporting and importing data</li><li>• Missing values and how to handle them with Python</li><li>• Processing different types of data with Python</li><li>• The NumPy package in Python for the creation of large, multi-dimensional arrays and matrices</li><li>• The Pandas library in Python for data analysis</li><li>• The Matplotlib library in Python for data visualization</li><li>• Main features and components of Hadoop framework</li><li>• The Hadoop Distributed File System</li><li>• Pig programming language</li></ul>

## 4 IoT Lab Resources

### 4.1 Requirements

In order to be able to do this challenge, you will need the following things:

- Python installed on your PC
- Internet access
- The FortiClient to make a VPN connection to the Hadoop cluster using the credentials you have received
- An SSH client (e.g. Putty) to make an SSH connection to the Hadoop cluster using the credentials you have received

## 5 The project step by step

To complete successfully the project work you can follow this step sequence

SEQUENCE	WORK	Estimated time
<b>STEP 1: Load the provided dataset and explore it</b>	<i>Students will import libraries and the given dataset into Python and will execute the proper code to inspect its data and structure (columns, data types, null and na values)</i>	<b>0.5h</b>
<b>STEP 2: Create a dataframe with selected columns and values</b>	<i>Students will manipulate the dataset (delete specified rows, check for unique values, delete columns), will display summary statistics of the data and will create a dataframe with numerical data</i>	<b>1h</b>
<b>STEP 3: Visualize the columns of the dataframe</b>	<i>Students will load the seaborn library and will use it along with matplotlib to create histogram and boxplot visualizations for each column of the dataset using a loop command iterating over the dataframe columns</i>	<b>1,5h</b>
<b>STEP 4: Setup of the VPN client and connection the Hadoop cluster</b>	<i>Students will install and configure the FortiClient to establish a VPN connection to the network at which the Hadoop cluster runs. Then using an SSH client they will make an SSH connection to the Hadoop cluster</i>	<b>0.5h</b>
<b>STEP 5: Connection to the container running Hadoop and uploading of data set at HDFS</b>	<i>Students will connect at the container running HDFS. They will create a personal folder at HDFS and upload the dataset</i>	<b>0.5h</b>
<b>STEP 6: Data analysis using Apache Pig</b>	<i>Students will make the required data analysis using Apache Pig</i>	<b>2h</b>

## 6 Expected result example

### Answers for the data processing with Python

This project is about processing data collected by a number of sensors at production machines. The answers we are asked to give, using the Python language, are the following:

1. Import the pandas, numpy and matplotlib libraries
2. Load the dataset (maintenance21.csv)
3. Display the first and the last 5 lines of the dataset.
4. Use the info() function to see columns, data type per column and number of non- null values count
5. Display the sum of null values per column
6. Calculate the percentage of missing values in each column (hint: write your own expression using isnull(), sum() and len() functions)
7. Delete all rows that contain na values
8. Confirm that the dataset has now less lines (971 rather than 1000)
9. Display all basic descriptive statistics for the dataset (count, mean, std, min, max and quartiles)
10. Display a statistics summary (like above) but now only for column with numerical datatypes, such as int, float
11. Check for the number of unique values in each column
12. Gather all numerical columns in a dataframe and name it num\_cols
13. Remove column «broken» from num\_cols
14. Use the seaborn library and use it along with matplotlib to create for each column a histogram and a boxplot visualization (hint: try using a for loop to cover each column in num\_cols)

The answers of these questions will be given in the next sections.

The answers to the first three queries are presented in the following screenshot.

```
#Import necessary Libraries
import pandas as pd
import numpy as np
import seaborn as sns
from matplotlib import pyplot as plt

#Load the dataset (maintenance_data.csv)
data = pd.read_csv('maintenance_data.csv')
```

```
#Display the first 5 Lines of the dataset
data.head(5)
```

	lifetime	broken	pressureInd	moistureInd	temperatureInd	team	provider
0	56.0	0.0	92.178854	104.230204	96.517159	TeamA	Provider4
1	81.0	1.0	72.075938	103.065701	87.271062	TeamC	Provider4
2	60.0	0.0	96.272254	77.801376	112.196170	TeamA	Provider1
3	86.0	1.0	94.406461	108.493608	72.025374	TeamC	Provider2
4	34.0	0.0	97.752899	99.413492	103.756271	TeamB	Provider1

```
#Display the the Last 5 Lines of the dataset
data.tail(5)
```

	lifetime	broken	pressureInd	moistureInd	temperatureInd	team	provider
995	88.0	1.0	88.589759	112.167556	99.861456	TeamB	Provider4
996	88.0	1.0	116.727075	110.871332	95.075631	TeamA	Provider4
997	22.0	0.0	104.026778	88.212873	83.221220	TeamB	Provider1
998	78.0	0.0	104.911649	104.257296	83.421491	NaN	Provider4
999	63.0	0.0	116.901354	99.998694	47.641493	TeamB	Provider1

As we can see, we firstly imported the necessary libraries, in order to do the data processing with Python. The second step involves the usage of the `read_csv()` function of Pandas, in order to load the data set that we are going to process. Afterwards, we use the `.head()` and `.tail()` functions, with the parameter of number «5», which prints the first five and the last five lines of the data set.

The answers to the fourth and fifth query are presented in the following screenshot.

```
#Use the info() function to see columns, data type per column and number of non-null values count
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 7 columns):
Column Non-Null Count Dtype
--- -
0 lifetime 999 non-null float64
1 broken 998 non-null float64
2 pressureInd 994 non-null float64
3 moistureInd 994 non-null float64
4 temperatureInd 992 non-null float64
5 team 994 non-null object
6 provider 999 non-null object
dtypes: float64(5), object(2)
memory usage: 54.8+ KB
```

```
#Display the sum of null values per column
data.isna().sum()
```

```
lifetime 1
broken 2
pressureInd 6
moistureInd 6
temperatureInd 8
team 6
provider 1
dtype: int64
```

As we can see, there is a usage of the .info() function in order to see the columns, their data types, and the non-null values of their data. Furthermore, we use .isna().sum() functions, which presents the number of null/na values per column. For example, the number of null values for the column «team» is 6.

The next queries are about the calculation of the percentage of null values, the removal of rows that contain null values, and the confirmation that the dataset has now 971 lines.

```
#Custom function to find the percentage of null values
def PercentNans(data, column):
 count = 0
 number = 0
 for thing in data[column]:
 if pd.isna(thing)==True:
 count+=1
 number+=1
 print("Percentage of Null Values for {}: {}".format(column, (count/len(data[column]))))

for col in data:
 PercentNans(data, col)

Percentage of Null Values for lifetime: 0.001%
Percentage of Null Values for broken: 0.002%
Percentage of Null Values for pressureInd: 0.006%
Percentage of Null Values for moistureInd: 0.006%
Percentage of Null Values for temperatureInd: 0.008%
Percentage of Null Values for team: 0.006%
Percentage of Null Values for provider: 0.001%

#Delete rows that contain na values
data = data.dropna()

data.isna().sum()

lifetime 0
broken 0
pressureInd 0
moistureInd 0
temperatureInd 0
team 6
provider 0
dtype: int64

#Confirm that the dataset has now 971 Lines
len(data)

971
```

The answer to the first matter is done by the creation of the «PercentNans» function, which is used to calculate the percent of null/na values in a column defined to it. This function is later called in a for-loop, and prints the percentages for every column of the data. In the previous screenshot we can also see the usage of .dropna() function, which deletes the rows that contain na values, as well as the confirmation that the data set has now 971 lines. The confirmation is done by the usage of len() function, which printed the length of the data, or in other words, the total number of the instances of the data set.

All the statistics for the dataset are presented by the simple .describe() function, as we can observe and in the following screenshot.



```
#Display all basic descriptive statistics for the dataset
data.describe()
```

	lifetime	broken	pressureInd	moistureInd	temperatureInd
count	971.000000	971.000000	971.000000	971.000000	971.000000
mean	55.019567	0.395469	98.384816	99.338883	100.658143
std	26.531262	0.489203	20.032275	9.946095	19.661525
min	1.000000	0.000000	33.481917	58.547301	42.279598
25%	33.500000	0.000000	85.310989	92.747920	87.794433
50%	60.000000	0.000000	96.978536	99.413137	100.623052
75%	80.000000	1.000000	112.121547	106.145002	113.709515
max	93.000000	1.000000	173.282541	128.595038	172.544140

The next query is about presenting all the statistics for just the numerical columns of the data set. For this to happen, a creation of a for-loop was made, which for every column of the data set checks if the column is not type of an object. If this is true, the result is that this column is a numeric one, and we then store its name in an array called «numericColumns». Finally, for every column name in this array, we use the .describe() function to check it's statistics.

```
#Display statistics summary but now only for column with numerical datatypes
numericColumns = []
for col in data: #For every column in the data we are processing
 if (data[col].dtype) != 'O': #If the column is not type of object (as a result, it's a numeric one)
 numericColumns.append(col) #Store it in numericColumns array

for col in numericColumns:
 print('Statistics summary for {} column:'.format(col))
 print(data[col].describe())
 print('\n')
```

Some of the results of these lines of code are presented in the following screenshot.

```
Statistics summary for lifetime column:
count 971.000000
mean 55.019567
std 26.531262
min 1.000000
25% 33.500000
50% 60.000000
75% 80.000000
max 93.000000
Name: lifetime, dtype: float64
```

```
Statistics summary for broken column:
count 971.000000
mean 0.395469
std 0.489203
min 0.000000
25% 0.000000
50% 0.000000
75% 1.000000
max 1.000000
Name: broken, dtype: float64
```

The next query is about finding the unique values in every column of the data set. For this to happen, a creation of a function called «findUnique» was made, which is later called in a for-loop, and returns the number of unique values for every column. Further details about these lines of code are presented in the following screenshot.

```
#Check for the number of unique values in each column
def findUnique(data):
 storageArray = []
 number = 0
 for value in data:
 if value in storageArray:
 pass
 else:
 storageArray.append(value)
 number+=1
 return number

print('Unique Values\n-----')
for col in data:
 number = findUnique(data[col])
 print('Column {}: {}'.format(col, number))
```

Unique Values  
-----  
Column lifetime: 90  
Column broken: 2  
Column pressureInd: 971  
Column moistureInd: 971  
Column temperatureInd: 971  
Column team: 3  
Column provider: 4

To go on, the next query asks to gather all the numerical columns in a dataframe and name it num\_cols. The Python code which answers this question is presented in the following screenshot.

```
#Gather all numerical columns in a dataframe and name it num_cols
num_cols = data.select_dtypes(include=np.number)
num_cols
```

	lifetime	broken	pressureInd	moistureInd	temperatureInd
0	56.0	0.0	92.178854	104.230204	96.517159
1	81.0	1.0	72.075938	103.065701	87.271062
2	60.0	0.0	96.272254	77.801376	112.196170
3	86.0	1.0	94.406461	108.493608	72.025374
4	34.0	0.0	97.752899	99.413492	103.756271
...	...	...	...	...	...
994	80.0	1.0	110.229747	101.445523	75.630577
995	88.0	1.0	88.589759	112.167556	99.861456
996	88.0	1.0	116.727075	110.871332	95.075631
997	22.0	0.0	104.026778	88.212873	83.221220
999	63.0	0.0	116.901354	99.998694	47.641493

As for the next step, we are asked to delete the column «broken» from the num\_cols data. This is achieved by the usage of .drop() function. The parameters that we gave to this function in order to do the task that we wanted is 1) the name of the column (['broken']) and 2) the axis (=1) that is going to be removed. Further details are presented in the following screenshot, which also presents the Python code for this query.

```
#Remove column "broken" from num_cols
num_cols = num_cols.drop(['broken'], axis=1)
num_cols
```

	lifetime	pressureInd	moistureInd	temperatureInd
0	56.0	92.178854	104.230204	96.517159
1	81.0	72.075938	103.065701	87.271062
2	60.0	96.272254	77.801376	112.196170
3	86.0	94.406461	108.493608	72.025374
4	34.0	97.752899	99.413492	103.756271
...	...	...	...	...
994	80.0	110.229747	101.445523	75.630577
995	88.0	88.589759	112.167556	99.861456
996	88.0	116.727075	110.871332	95.075631
997	22.0	104.026778	88.212873	83.221220
999	63.0	116.901354	99.998694	47.641493

The final query was about using the Seaborn and Matplotlib libraries to create for each column a histogram and a boxplot. For these, we created two functions, the «boxPlotCostum» and the «histPlotCostum». The first one, uses Seaborn in order to present a boxplot for some data defined to it. The second one, uses Matplotlib, and more specifically the Pyplot module, in order to create a histogram for data defined to it. Both of these functions are used in a for-loop and print the visualizations for every column in num\_cols data. These lines of code are presented in the following screenshot:

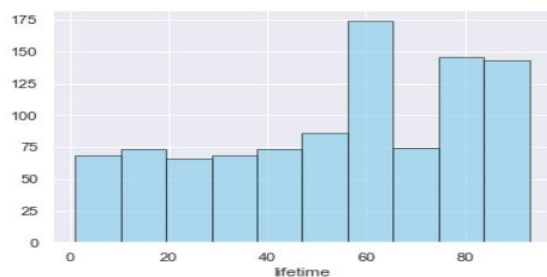
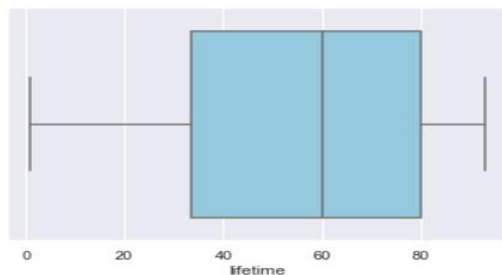
```
#Use seaborn and matplotlib to create for each column a histogram and a boxplot visualization
def boxPlotCostum(data):
 sns.boxplot(x = data, color = 'skyblue')
 plt.show()

def histPlotCostum(data, value):
 plt.hist(data, color='skyblue', edgecolor='k', alpha=0.65)
 plt.xlabel("{}".format(value))
 plt.show()

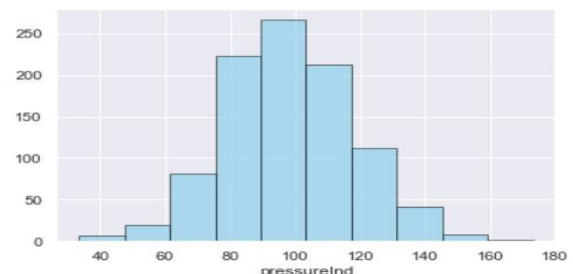
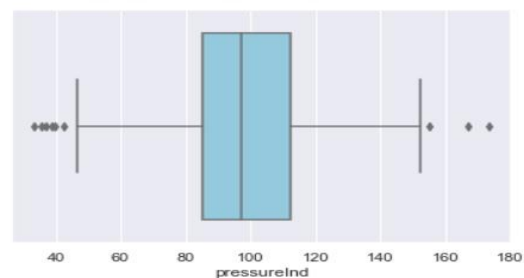
for col in num_cols:
 print("Plots for {} column\n-----".format(col))
 boxPlotCostum(num_cols[col])
 histPlotCostum(num_cols[col], col)
```

Some of the plots created from these lines of code are presented in the following screenshot.

Plots for lifetime column



Plots for pressureInd column



## Answers for the data processing with Apache Pig

As a next step, we are asked to process the dataset and make an analysis to it using Apache Pig. The queries we are asked to do are the following:

15. Select the machines with temperature bigger than 90
16. Select the machine coming from provider 4
17. Get the temperature, pressure and moisture from all machines
18. Get the temperature, pressure and moisture from all machines coming from provider 4



19. Split the initial data set into 2 different datasets, one containing the records from machines with moisture bigger than 20 and another containing the records from machine with temperature less or equal to 100

The answers of these questions will be given in the following sections.

Hadoop was installed inside a virtual machine, which was running Ubuntu 22.04.02 LTS. The folder that Hadoop was installed is under the home and the user «konskout» folder. The Hadoop version that was installed was Hadoop 2.7.3. In the same folder of Hadoop, there was also installed Apache Pig. To start the necessary Hadoop functions in the localhost, as well as to also start Pig, there were a number of simple CMD commands followed, as we can observe in the following screenshot.

```
konskout@DTAM-PC: ~/hadoop/hadoop-2.7.3/sbin
konskout@DTAM-PC: $ cd ~/hadoop/hadoop-2.7.3/sbin
konskout@DTAM-PC:~/hadoop/hadoop-2.7.3/sbin$./start-all.sh
This script is deprecated. Instead use start-dfs.sh and start-yarn.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /home/konskout/hadoop/hadoop-2.7.3/logs/hadoop-konskout-namenode-DTAM-PC.out
localhost: starting datanode, logging to /home/konskout/hadoop/hadoop-2.7.3/logs/hadoop-konskout-datanode-DTAM-PC.out
Starting secondary namenodes [0.0.0.0]
0.0.0.0: starting secondarynamenode, logging to /home/konskout/hadoop/hadoop-2.7.3/logs/hadoop-konskout-secondarynamenode-DTAM-PC.out
starting yarn daemons
starting resourcemanager, logging to /home/konskout/hadoop/hadoop-2.7.3/logs/yarn-konskout-resourcemanager-DTAM-PC.out
localhost: starting nodemanager, logging to /home/konskout/hadoop/hadoop-2.7.3/logs/yarn-konskout-nodemanager-DTAM-PC.out
konskout@DTAM-PC:~/hadoop/hadoop-2.7.3/sbin$ jps
2853 ResourceManager
2550 DataNode
3258 Jps
2717 SecondaryNameNode
2974 NodeManager
konskout@DTAM-PC:~/hadoop/hadoop-2.7.3/sbin$ pig -x local
23/05/03 15:15:16 INFO pig.ExecTypeProvider: Trying ExecType : LOCAL
23/05/03 15:15:16 INFO pig.ExecTypeProvider: Picked LOCAL as the ExecType
2023-05-03 15:15:16,073 [main] INFO org.apache.pig.Main - Apache Pig version 0.17.0 (r1797386) compiled Jun 02 2017, 15:41:58
2023-05-03 15:15:16,073 [main] INFO org.apache.pig.Main - Logging error messages to: /home/konskout/hadoop/hadoop-2.7.3/sbin/pig_1683116116071.log
2023-05-03 15:15:16,122 [main] INFO org.apache.pig.impl.util.Utils - Default bootup file /home/konskout/.pigbootup not found
2023-05-03 15:15:16,320 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - mapred.job.tracker is deprecated. Instead, use mapreduce.jobtracker.address
2023-05-03 15:15:16,322 [main] INFO org.apache.pig.backend.hadoop.executionengine.HExecutionEngine - Connecting to hadoop file system at: file:///
2023-05-03 15:15:16,487 [main] INFO org.apache.hadoop.conf.Configuration.deprecation - io.bytes.per.checksum is deprecated. Instead, use dfs.bytes-per-checksum
2023-05-03 15:15:16,502 [main] INFO org.apache.pig.PigServer - Pig Script ID for the session: PIG-default-8423168e-abff-473e-9cc3-cc3c428e3eaf
2023-05-03 15:15:16,503 [main] WARN org.apache.pig.PigServer - ATS is disabled since yarn.timeline-service.enabled set to false
```

The next step was to load the data set that we would processed in Apache Pig. This happened with the usage of the Pig command presented in the following screenshot.

```
grunt> data = LOAD '/home/konskout/Documents/maintenance_data.csv' using PigStorage(',') AS (lifetime:int, broken:int, pressure:double, moisture:double, temp: double, team:chararray, provider:chararray);
```

This Pig command that was used was the:

```
“ data = LOAD '/home/konskout/Documents/maintenance_data.csv' using
PigStorage(',') AS (lifetime:int, broken:int, pressure:double, moisture:double, temp:
double, team:chararray, provider:chararray); “
```

In order to confirm that the data set was loaded correctly, we used the “DUMP data;” command. This command printed all the instances of the data set that were loaded from the maintenance\_data CSV file.

```
(30,0,84.58244428,98.42001087,88.00175655,TeamC,Provider1)
(79,1,82.61501033,102.9596453,124.0469975,TeamB,Provider1)
(44,0,104.4124785,104.8934097,117.2735726,TeamB,Provider3)
(80,1,70.76412566,88.90264367,107.2609915,TeamA,Provider1)
(88,1,99.68203559,115.8417289,104.0767926,TeamA,Provider4)
(88,1,74.03205981,81.48540424,122.9497116,TeamB,Provider4)
(48,0,143.5503382,97.50586632,121.1105841,TeamC,Provider1)
(81,1,77.69900009,109.9499969,143.9519908,TeamC,Provider4)
(81,1,79.74938628,78.281154,87.93739852,TeamC,Provider4)
(81,0,90.78063751,99.8653775,99.6011896,TeamC,Provider2)
(60,1,117.7069331,114.818523,104.29978,TeamC,Provider3)
(64,0,119.6780052,99.67938347,114.9546663,TeamA,Provider4)
(88,1,70.16802889,93.12458318,86.05930834,TeamB,Provider4)
(86,1,83.27692515,109.7798078,,TeamC,Provider2)
(80,1,111.9914415,106.2575243,110.3341415,TeamB,Provider1)
(65,1,102.7767557,111.4517798,112.9557703,TeamA,Provider3)
(55,0,87.946098,96.13748874,107.1117844,TeamA,Provider2)
(25,0,92.57385892,103.6563739,103.262856,TeamA,Provider2)
(52,0,88.20535721,110.3469315,99.49648475,TeamC,Provider4)
(29,0,140.3729067,104.5390305,82.25351215,TeamA,Provider1)
(58,0,95.30752801,97.49013516,102.3134457,TeamA,Provider3)
(66,1,76.76990774,110.1044455,76.71751602,TeamB,Provider3)
(36,0,119.0339468,120.1119323,70.32511371,TeamA,Provider1)
(88,1,113.7665009,88.42367648,110.0930537,TeamB,Provider4)
(67,0,109.7687115,93.37811705,122.1599585,TeamB,Provider2)
(45,0,125.7018715,101.9967219,58.30095817,TeamA,Provider2)
(30,0,93.88255716,103.1290149,110.2320489,TeamB,Provider3)
(65,1,86.98757582,123.6561241,106.6508842,TeamA,Provider3)
(76,0,87.85255835,94.28668724,114.1031517,TeamC,Provider2)
(60,1,115.7507865,98.76222896,101.2066501,TeamC,Provider3)
(65,1,94.68396594,108.9614215,118.0273944,TeamA,Provider3)
(80,1,110.2297466,101.4455226,75.63057702,TeamB,Provider1)
(88,1,88.58975909,112.1675561,99.86145565,TeamB,Provider4)
(88,1,116.7270751,110.8713319,95.07563134,TeamA,Provider4)
(22,0,104.0267784,88.21287267,83.22122036,TeamB,Provider1)
(78,0,104.9116487,104.2572957,83.42149109,,Provider4)
(63,0,116.9013541,99.99869359,47.64149344,TeamB,Provider1)
grunt>
```

The first query of the Apache Pig tasks is about selecting the machines with temperature bigger than 90. The answer is presented in the following screenshot.

```
grunt> highTemps = FILTER data BY (temp>90);
2023-05-03 15:24:11,212 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 1 time(s).
grunt>
```

The Pig command that was used is the:

“highTemps = FILTER data BY (temp>90);”.

The data with machines with high temperatures that were filtered from the initial data can be seen with the usage of the “DUMP highTemps;” command. This command presents the following:



```
(68,0,118.0070529,84.5964103,109.636018,TeamA,Provider2)
(74,1,110.1976712,109.9128786,96.71791287,TeamC,Provider1)
(27,0,80.19902714,108.4494863,96.28547323,TeamA,Provider3)
(13,0,100.0915524,98.56383772,127.8021082,TeamA,Provider3)
(49,0,105.5421334,97.28968823,112.9373963,TeamB,Provider4)
(2,0,70.16567395,101.9694427,114.6269398,TeamC,Provider3)
(80,1,75.6867736,97.27128211,112.4831253,TeamB,Provider1)
(18,0,150.1404798,94.52318723,128.2809537,TeamC,Provider3)
(23,0,107.5213703,98.97375537,110.1142502,TeamB,Provider2)
(28,0,56.72643095,92.31284639,117.2177177,TeamC,Provider2)
(65,1,81.71098194,99.14965123,93.16406456,TeamB,Provider3)
(88,1,103.2401042,106.61584,90.70206847,TeamB,Provider4)
(79,1,82.61501033,102.9596453,124.0469975,TeamB,Provider1)
(44,0,104.4124785,104.8934097,117.2735726,TeamB,Provider3)
(80,1,70.76412566,88.90264367,107.2609915,TeamA,Provider1)
(88,1,99.68203559,115.8417289,104.0767926,TeamA,Provider4)
(88,1,74.03205981,81.48540424,122.9497116,TeamB,Provider4)
(48,0,143.5503382,97.50586632,121.1105841,TeamC,Provider1)
(81,1,77.69900009,109.9499969,143.9519908,TeamC,Provider4)
(81,0,90.78063751,99.8653775,99.6011896,TeamC,Provider2)
(60,1,117.7069331,114.818523,104.29978,TeamC,Provider3)
(64,0,119.6780052,99.67938347,114.9546663,TeamA,Provider4)
(80,1,111.9914415,106.2575243,110.3341415,TeamB,Provider1)
(65,1,102.7767557,111.4517798,112.9557703,TeamA,Provider3)
(55,0,87.946098,96.13748874,107.1117844,TeamA,Provider2)
(25,0,92.57385892,103.6563739,103.262856,TeamA,Provider2)
(52,0,88.20535721,110.3469315,99.49648475,TeamC,Provider4)
(58,0,95.30752801,97.49013516,102.3134457,TeamA,Provider3)
(88,1,113.7665009,88.42367648,110.0930537,TeamB,Provider4)
(67,0,109.7687115,93.37811705,122.1599585,TeamB,Provider2)
(30,0,93.88255716,103.1290149,110.2320489,TeamB,Provider3)
(65,1,86.98757582,123.6561241,106.6508842,TeamA,Provider3)
(76,0,87.85255835,94.28668724,114.1031517,TeamC,Provider2)
(60,1,115.7507865,98.76222896,101.2066501,TeamC,Provider3)
(65,1,94.68396594,108.9614215,118.0273944,TeamA,Provider3)
(88,1,88.58975909,112.1675561,99.86145565,TeamB,Provider4)
(88,1,116.7270751,110.8713319,95.07563134,TeamA,Provider4)
grunt>
```

The second query is about selecting machines coming from provider 4. The answer is presented in the following screenshot.

```
grunt> filtered_Prov4 = FILTER data BY (provider == 'Provider4');
grunt>
```

The Pig command that was used is the:

“filtered\_Prov4 = FILTER data BY (provider == 'Provider4');”.

The data with machines coming from provider 4 can be seen with the usage of “DUMP filtered\_Prov4;” command. The results are the following:

```
(88,1,101.8850304,80.20631364,111.2401286,TeamA,Provider4)
(81,1,52.49765001,90.09954474,84.82670487,TeamC,Provider4)
(73,0,103.8813764,93.1331722,92.27922542,TeamA,Provider4)
(88,1,84.29478616,85.65029662,93.81460506,TeamB,Provider4)
(77,0,98.14966037,84.2348114,99.79866988,TeamC,Provider4)
(88,1,120.6151502,86.09934693,89.08310201,TeamB,Provider4)
(49,0,105.5421334,97.28968823,112.9373963,TeamB,Provider4)
(88,1,103.2401042,106.61584,90.70206847,TeamB,Provider4)
(88,1,99.68203559,115.8417289,104.0767926,TeamA,Provider4)
(88,1,74.03205981,81.48540424,122.9497116,TeamB,Provider4)
(81,1,77.69900009,109.9499969,143.9519908,TeamC,Provider4)
(81,1,79.74938628,78.281154,87.93739852,TeamC,Provider4)
(64,0,119.6780052,99.67938347,114.9546663,TeamA,Provider4)
(88,1,70.16802889,93.12458318,86.05930834,TeamB,Provider4)
(52,0,88.20535721,110.3469315,99.49648475,TeamC,Provider4)
(88,1,113.7665009,88.42367648,110.0930537,TeamB,Provider4)
(88,1,88.58975909,112.1675561,99.86145565,TeamB,Provider4)
(88,1,116.7270751,110.8713319,95.07563134,TeamA,Provider4)
(78,0,104.9116487,104.2572957,83.42149109,,Provider4)
grunt>
```

The third query is about getting the temperature, pressure and moisture from all machines.

The Pig command that was used for this, is the following:

```
grunt> filtered_TPM = FOREACH data GENERATE pressure, moisture, temp;
2023-05-03 15:28:30,359 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 1 time(s).
grunt>
```

The command that was used is the:

“filtered\_TPM = FOREACH data GENERATE pressure, moisture, temp;”.

These data filtered can be seen with the: “DUMP filtered\_TPM;” command. The results are presented in the following screenshot.

```
(84.58244428,98.42001087,88.00175655)
(82.61501033,102.9596453,124.0469975)
(104.4124785,104.8934097,117.2735726)
(70.76412566,88.90264367,107.2609915)
(99.68203559,115.8417289,104.0767926)
(74.03205981,81.48540424,122.9497116)
(143.5503382,97.50586632,121.1105841)
(77.69900009,109.9499969,143.9519908)
(79.74938628,78.281154,87.93739852)
(90.78063751,99.8653775,99.6011896)
(117.7069331,114.818523,104.29978)
(119.6780052,99.67938347,114.9546663)
(70.16802889,93.12458318,86.05930834)
(83.27692515,109.7798078,)
(111.9914415,106.2575243,110.3341415)
(102.7767557,111.4517798,112.9557703)
(87.946098,96.13748874,107.1117844)
(92.57385892,103.6563739,103.262856)
(88.20535721,110.3469315,99.49648475)
(140.3729067,104.5390305,82.25351215)
(95.30752801,97.49013516,102.3134457)
(76.76990774,110.1044455,76.71751602)
(119.0339468,120.1119323,70.32511371)
(113.7665009,88.42367648,110.0930537)
(109.7687115,93.37811705,122.1599585)
(125.7018715,101.9967219,58.30095817)
(93.88255716,103.1290149,110.2320489)
(86.98757582,123.6561241,106.6508842)
(77.85255835,94.28668724,114.1031517)
(115.7507865,98.76222896,101.2066501)
(94.68396594,108.9614215,118.0273944)
(110.2297466,101.4455226,75.63057702)
(88.58975909,112.1675561,99.86145565)
(116.7270751,110.8713319,95.07563134)
(104.0267784,88.21287267,83.22122036)
(104.9116487,104.2572957,83.42149109)
(116.9013541,99.99869359,47.64149344)
grunt>
```

The fourth query is about creating a script which selects the pressure, the temperature, and the moisture from machines coming from provider 4. This is done with the commands which are presented in the following screenshot.

```
grunt> filtered_Prov4 = FILTER data BY (provider == 'Provider4');
2023-05-03 15:30:29,177 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 1 time(s).
grunt> filtered_Prov4_2 = FOREACH filtered_Prov4 GENERATE pressure, moisture, temp;
2023-05-03 15:30:45,980 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 1 time(s).
grunt>
```

The commands that were used are the:

“filtered\_Prov4 = FILTER data BY (provider == 'Provider4');”

filtered\_Prov4\_2 = FOREACH filtered\_Prov4 GENERATE pressure, moisture, temp;”.

The data generated can be seen with the: “DUMP filtered\_Prov4\_2;” command. The results are presented in the following screenshot.



```
(93.78066682,95.35328972,82.56648328)
(98.83673629,104.9088072,114.3900269)
(101.6701795,82.07504486,102.3655284)
(86.88757075,106.6893263,95.55979966)
(98.68886709,125.3812758,123.3805805)
(101.637611,97.60178301,118.0431521)
(101.8850304,80.20631364,111.2401286)
(52.49765001,90.09954474,84.82670487)
(103.8813764,93.1331722,92.27922542)
(84.29478616,85.65029662,93.81460506)
(98.14966037,84.2348114,99.79866988)
(120.6151502,86.09934693,89.08310201)
(105.5421334,97.28968823,112.9373963)
(103.2401042,106.61584,90.70206847)
(99.68203559,115.8417289,104.0767926)
(74.03205981,81.48540424,122.9497116)
(77.69900009,109.9499969,143.9519908)
(79.74938628,78.281154,87.93739852)
(119.6780052,99.67938347,114.9546663)
(70.16802889,93.12458318,86.05930834)
(88.20535721,110.3469315,99.49648475)
(113.7665009,88.42367648,110.0930537)
(88.58975909,112.1675561,99.86145565)
(116.7270751,110.8713319,95.07563134)
(104.9116487,104.2572957,83.42149109)
grunt>
```

Finally, the last query for this project has to do with splitting the initial data set into 2 different data sets, one containing the records from machines with moisture bigger than 20 and another containing the records from machine with temperature less or equal to 100. This is done with the command which is presented in the following screenshot.

```
grunt> SPLIT data INTO moisture20 if moisture>20, temp100 if temp<=100;
2023-05-03 15:37:38,586 [main] WARN org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_DOUBLE 3 time(s).
grunt>
```

The command that was used for this is the:

“SPLIT data INTO moisture20 if moisture>20, temp100 if temp<=100;”.

The data generated where split in two parts, the first was named moisture20, and the other

temp100. In order to see the results we can use the following Pig expressions: “DUMP moisture20;” and “DUMP temp100;”. The outcomes of these commands are presented in the following screenshots.



```
(30,0,84.58244428,98.42001087,88.00175655,TeamC,Provider1)
(79,1,82.61501033,102.9596453,124.0469975,TeamB,Provider1)
(44,0,104.4124785,104.8934097,117.2735726,TeamB,Provider3)
(80,1,70.76412566,88.90264367,107.2609915,TeamA,Provider1)
(88,1,99.68203559,115.8417289,104.0767926,TeamA,Provider4)
(88,1,74.03205981,81.48540424,122.9497116,TeamB,Provider4)
(48,0,143.5503382,97.50586632,121.1105841,TeamC,Provider1)
(81,1,77.69900009,109.9499969,143.9519908,TeamC,Provider4)
(81,1,79.74938628,78.281154,87.93739852,TeamC,Provider4)
(81,0,90.78063751,99.8653775,99.6011896,TeamC,Provider2)
(60,1,117.7069331,114.818523,104.29978,TeamC,Provider3)
(64,0,119.6780052,99.67938347,114.9546663,TeamA,Provider4)
(88,1,70.16802889,93.12458318,86.05930834,TeamB,Provider4)
(86,1,83.27692515,109.7798078,,TeamC,Provider2)
(80,1,111.9914415,106.2575243,110.3341415,TeamB,Provider1)
(65,1,102.7767557,111.4517798,112.9557703,TeamA,Provider3)
(55,0,87.946098,96.13748874,107.1117844,TeamA,Provider2)
(25,0,92.57385892,103.6563739,103.262856,TeamA,Provider2)
(52,0,88.20535721,110.3469315,99.49648475,TeamC,Provider4)
(29,0,140.3729067,104.5390305,82.25351215,TeamA,Provider1)
(58,0,95.30752801,97.49013516,102.3134457,TeamA,Provider3)
(66,1,76.76990774,110.1044455,76.71751602,TeamB,Provider3)
(36,0,119.0339468,120.1119323,70.32511371,TeamA,Provider1)
(88,1,113.7665009,88.42367648,110.0930537,TeamB,Provider4)
(67,0,109.7687115,93.37811705,122.1599585,TeamB,Provider2)
(45,0,125.7018715,101.9967219,58.30095817,TeamA,Provider2)
(30,0,93.88255716,103.1290149,110.2320489,TeamB,Provider3)
(65,1,86.98757582,123.6561241,106.6508842,TeamA,Provider3)
(76,0,87.85255835,94.28668724,114.1031517,TeamC,Provider2)
(60,1,115.7507865,98.76222896,101.2066501,TeamC,Provider3)
(65,1,94.68396594,108.9614215,118.0273944,TeamA,Provider3)
(80,1,110.2297466,101.4455226,75.63057702,TeamB,Provider1)
(88,1,88.58975909,112.1675561,99.86145565,TeamB,Provider4)
(88,1,116.7270751,110.8713319,95.07563134,TeamA,Provider4)
(22,0,104.0267784,88.21287267,83.22122036,TeamB,Provider1)
(78,0,104.9116487,104.2572957,83.42149109,,Provider4)
(63,0,116.9013541,99.99869359,47.64149344,TeamB,Provider1)
grunt>
```

```
(10,0,133.4265573,107.4336744,95.73557981,TeamA,Provider2)
(42,0,88.40955072,111.8542572,92.50320389,TeamC,Provider2)
(12,0,69.80779035,83.67069847,82.23645826,TeamA,Provider2)
(80,1,129.1863512,90.99237878,91.12806915,TeamB,Provider1)
(57,0,112.9292352,97.33062442,96.27540641,TeamB,Provider2)
(44,0,79.59452653,100.1106956,98.43332131,TeamB,Provider3)
(9,0,90.78568329,94.05537051,84.73797429,TeamC,Provider1)
(81,1,52.49765001,90.09954474,84.82670487,TeamC,Provider4)
(65,1,119.0011817,108.5672605,90.21907271,TeamA,Provider3)
(73,0,103.8813764,93.1331722,92.27922542,TeamA,Provider4)
(88,1,84.29478616,85.65029662,93.81460506,TeamB,Provider4)
(66,1,71.92183381,106.8901076,66.868012,TeamB,Provider3)
(77,0,98.14966037,84.2348114,99.79866988,TeamC,Provider4)
(93,1,98.37929005,96.96697723,76.26973887,TeamA,Provider2)
(88,1,120.6151502,86.09934693,89.08310201,TeamB,Provider4)
(74,1,110.1976712,109.9128786,96.71791287,TeamC,Provider1)
(27,0,80.19902714,108.4494863,96.28547323,TeamA,Provider3)
(14,0,85.73088376,104.2604324,81.1547673,TeamA,Provider3)
(30,0,99.65564682,95.30300146,60.70238138,TeamC,Provider2)
(65,1,81.71098194,99.14965123,93.16406456,TeamB,Provider3)
(34,0,102.0595059,83.97148642,86.24532972,TeamC,Provider3)
(88,1,103.2401042,106.61584,90.70206847,TeamB,Provider4)
(30,0,84.58244428,98.42001087,88.00175655,TeamC,Provider1)
(81,1,79.74938628,78.281154,87.93739852,TeamC,Provider4)
(81,0,90.78063751,99.8653775,99.6011896,TeamC,Provider2)
(88,1,70.16802889,93.12458318,86.05930834,TeamB,Provider4)
(52,0,88.20535721,110.3469315,99.49648475,TeamC,Provider4)
(29,0,140.3729067,104.5390305,82.25351215,TeamA,Provider1)
(66,1,76.76990774,110.1044455,76.71751602,TeamB,Provider3)
(36,0,119.0339468,120.1119323,70.32511371,TeamA,Provider1)
(45,0,125.7018715,101.9967219,58.30095817,TeamA,Provider2)
(80,1,110.2297466,101.4455226,75.63057702,TeamB,Provider1)
(88,1,88.58975909,112.1675561,99.86145565,TeamB,Provider4)
(88,1,116.7270751,110.8713319,95.07563134,TeamA,Provider4)
(22,0,104.0267784,88.21287267,83.22122036,TeamB,Provider1)
(78,0,104.9116487,104.2572957,83.42149109,,Provider4)
(63,0,116.9013541,99.99869359,47.64149344,TeamB,Provider1)
grunt>
```





Co-funded by the  
Erasmus+ Programme  
of the European Union

