# <u>Healthcare Data Pipeline</u>

Project report submitted in partial fulfillment of the requirement for the degree of Bachelor of Technology

in

**Computer Science and Engineering/Information Technology**

By

Shivam Dabral (191273)

Under the supervision of

Dr. Rajni Mohana

to

Department of Computer Science & Engineering and Information Technology

**Jaypee University of Information Technology Waknaghat, Solan-173234, Himachal Pradesh**

# Candidate's Declaration

I hereby declare that the work presented in this report entitled **"Healthcare Data Pipeline"** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering/Information Technology** submitted in the department of Computer Science & Engineering and Information Technology**,** Jaypee University of Information Technology Waknaghat is an authentic record of my own work carried out over a period from February 2023 to May 2023 under the supervision of **Dr. Rajni Mohana** (Associate Professor in the Department of Computer Science and Engineering).
The matter embodied in the report has not been submitted for the award of any other degree or diploma.

(Student Signature)
Shivam Dabral, 191273.

This is to certify that the above statement made by the candidate is true to the best of my knowledge.

(Supervisor Signature)
Dr. Rajni Mohana
Associate Professor
Computer Science and Engineering
Dated

## JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY, WAKNAGHAT

## PLAGIARISM VERIFICATION REPORT

**Date: ……………………….**

**Type of Document (Tick): PhD Thesis | M.Tech Dissertation/ Report | B.Tech Project Report | Paper**

**Name: ________________________ __Department: _______________ Enrolment No ________**

**Contact No. ____________________________E-mail. ___________________________________**

**Name of the Supervisor: ____________________________________________________________**

**Title of the Thesis/Dissertation/Project Report/Paper (In Capital letters): ______________________**

________________________________________________________________________________

________________________________________________________________________________

### UNDERTAKING

I undertake that I am aware of the plagiarism related norms/ regulations, if I found guilty of any plagiarism and copyright violations in the above thesis/report even after award of degree, the University reserves the rights to withdraw/revoke my degree/report. Kindly allow me to avail Plagiarism verification report for the document mentioned above.

**Complete Thesis/Report Pages Detail:**

– Total No. of Pages =
– Total No. of Preliminary pages =
– Total No. of pages accommodate bibliography/references =

**(Signature of Student)**

### FOR DEPARTMENT USE

We have checked the thesis/report as per norms and found **Similarity Index** at ……………….(%). Therefore, we are forwarding the complete thesis/report for final plagiarism check. The plagiarism verification report may be handed over to the candidate.

**(Signature of Guide/Supervisor)** **Signature of HOD**

### FOR LRC USE

The above document was scanned for plagiarism check. The outcome of the same is reported below:

| **Copy Received on** | **Excluded** | **Similarity Index (%)** | **Generated Plagiarism Report Details** (Title, Abstract & Chapters) | |
|---|---|---|---|---|
| | • All Preliminary Pages<br>• Bibliography/Images/Quotes<br>• 14 Words String | | Word Counts | |
| **Report Generated on** | | | Character Counts | |
| | | **Submission ID** | Total Pages Scanned | |
| | | | File Size | |

**Checked by**
**Name & Signature** **Librarian**

……………………………………………………………………………………………………………

**Please send your complete thesis/report in (PDF) with Title Page, Abstract and Chapters in (Word File) through the supervisor at plagcheck.juit@gmail.com**

# ACKNOWLEDGEMENT

Firstly, I express my gratitude to the god who provided me with courage and fortitude to
complete the project.

I am grateful and wish my profound indebtedness to Supervisor Dr.Rajni Mohana, Associate Professor, Department of CSE Jaypee University of Information Technology, Wakhnaghat.

Deep Knowledge & keen interest of my supervisor in the field of "Data Processing" to carry out this Project. Her endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stages have made it possible to complete this Project.

I would like to express my heartiest gratitude to Dr. Rajni Mohana, Department of CSE, for her kind help to finish my Project.

I would also generously welcome each one of those individuals who have helped me straightforwardly or in a roundabout way in making this project a success. In this unique situation, I might want to thank the various staff individuals, both educating and non-instructing, which have developed their convenient help and facilitated my undertaking.

Finally, I must acknowledge with due respect the constant support and patience of my
parents.

(Student Signature)
Project Group No.: 94
Student Name: Shivam Dabral
Rollno.: 191273

# Contents

## LIST OF FIGURES

## LIST OF GRAPHS

# ABSTRACT

Big Data Processing is a matter of interest for many companies around the globe as they try to harness the true power of data. Similarly Nference labs private limited is trying to make use of healthcare data to provide people with better medical support. This project aims at exploring such various techniques that employ engines and frameworks that can generate useful data from raw data effectively and efficiently. Various techniques were examined based upon many research papers and compared. The results suggested the use of Apache Spark as an engine for computation. The data files were stored in parquet format with snappy compression, so that data occupies less space. Hence the aim was to come up with an efficient data generation pipeline that can handle Terabytes of data.

# CHAPTER 1 : INTRODUCTION

## 1.1 Introduction

In today's world there is an abundance of data, owing to the great advancement in computational power of computers, easy access to the internet and population explosion. In the last century only the top companies could afford harnessing huge amounts of data, but nowadays even individuals can not afford the data. With the abundance of data there was an advent of Big Data Processing. Big Data. Big Data processing refers to systematic collection and warehousing of data so as to harness its true potential.

Big tech companies can use the readily available data for making their goods more "marketable" by being more aware of the trend of the market,what people want, monitoring their shopping habits, etc. The companies also use data in the healthcare sector to train machines to diagnose diseases and to come up with a hypothesis depending upon results of various tests. Not just in healthcare, machines can be trained in various other sectors using the abundant data like to train self-driving cars, make recommendation systems for the consumers, and many more.

However, this data is ever growing and needs to be managed properly, otherwise it can incur huge costs for storing data and processing which also consumes a lot of time. Hence it is essential to maintain this data properly so that its powers can be used optimally for the good of the company.

## 1.2 Problem Statement

The issue with Big Data is explained by 5 Vs- Volume ( How big is the data?), Velocity (How frequent is the data appended?), Variety (Formats of data like image, audio, etc), Value (How much of the data stored is actually useful) and Veracity (How accurate the data is?). Dealing with these problems of big data the most useful step hence are data extraction, data transformation and the subsequent data loading. Data Extraction refers to extracting the data from sources while ensuring its accuracy(no or minimum outliers). Data can thus become too much to handle, however is it all necessary? Hence Data Transformation is required to remove the bulk which is of no good. And at last is Data Loading into a platform in a manner that it occupies less space and is easy to access. Also the data read speed must be fast to ensure less time spent in data processing. Hence making a chain of operations is the task that ensures all these principles are taken care of making use of data easy for the company and ensuring its utilization to the maximum.

## 1.3 Objectives

The project objectives are as follows:

- Create a pipeline of successive functions that can be used in data extraction, data transformation and data loading, in the specified order.
- Data Extraction should merge data which is different/inconsistent and received from different sources.
- Data Transformation must be efficient, capable of removing data bulk that is not required and can be dropped because of no utility. Thereby saving storage space and shall be cost effective also.
- Data Loading must be such that data is quickly accessible across all platforms like cloud. Also data must be stored in a format such that data retrieval is quick. It must adopt some compression algorithm so that it occupies the minimum space possible.
- Not only an effective but also an efficient data pipeline is required to ensure data is processed quickly.

## 1.4 Methodology

The project work first of all includes recognizing all utilities that would be required to accomplish the aforementioned tasks. Having chosen the utilities it is important to get familiar with them and adopt best programming practices to ensure effective and efficient data processing. This shall include using the Apache Spark engine for processing  huge amounts of data. Being familiar with Python using PySpark, a python api for Apache Spark is necessary. Also getting used to Apache Airflow for managing the pipeline is necessary. Whole pipeline needs to be automated as each process involved may take hours, hence any manual triggers must be avoided to maintain the flow of the process and also save cost by not allowing clusters not to sit idle.

## 1.5 Organization

Chapter 2: Includes literature Survey containing studies, analysis and brief report for various research papers on data processing  and information available in relevant journals, conference papers etc.

Chapter 3: Includes the system development which contains a brief note on various utilities studied for accomplishing this task. Also it contains a detailed report on how the project was made and finished.

Chapter 4: Includes the results of the pipeline made and a small performance review of how it was performed.

Chapter 5: The conclusion is contained here which includes a discussion upon the results. Also this part houses a critical analysis of what has been done and what better can be done to improve it further i.e., the future developments.

# CHAPTER 2 : LITERATURE SURVEY

"Distributed Stream Processing Frameworks: A Comparative Study" by Li et al. (2018).[1] This paper compares four popular distributed stream processing frameworks, including Apache Spark Streaming, Apache Flink, Apache Storm, and Apache Samza. The authors evaluate the performance, scalability, and fault-tolerance of each framework, and provide insights into the strengths and weaknesses of each tool. So basically, this paper claims Flink to have least latency, Samza to have least CPU usage and in the end Spark to have least disk consumption among all. This study used Twitter Api and Kafka to fetch tweets and then transformed them before loading them into ElasticSearch.

"A Survey of Big Data Processing Frameworks in Healthcare Sector" by Saravanan et al. (2018) is another research paper that presents a comprehensive study of various frameworks for big data processing in the healthcare sector.[2] The study compares Apache Spark, Apache Hadoop and Apache Storm for big data processing, discussing their pros and cons. Apache Spark is a popular big data processing framework that offers high performance and scalability, making it well-suited for healthcare applications that require real-time data processing and analysis and therefore appears above the two in this sector where privacy and security are also a huge concern.

Kumar et al. (2017)'s research article "Big Data Processing Using Apache Spark in Hadoop Environment" analyses the performance and efficiency of Apache Spark for large data processing in a Hadoop system. The authors compare Spark's performance to that of other large data processing frameworks, such as Apache Hadoop and Apache Storm, and show that Spark exceeds them in terms of processing speed and efficiency.[3] Basically the trials also look at how different Spark parameters, such as the number of partitions, affect processing performance. The authors discover that increasing the number of partitions can enhance processing performance, but only up to a certain limit.

The foundational research article "Apache Spark: A Unified Engine for Big Data Processing" by Zaharia et al. (2016) introduces Apache Spark and outlines its architecture and main features.[4] Basically the purpose of this article is to present an in-depth analysis of Spark's design and to emphasise its benefits over alternative large data processing frameworks.The authors begin by analysing the constraints of existing large data processing systems such as Hadoop MapReduce, as well as the need for a unified engine capable of handling a broad range of workloads such as batch processing, interactive queries, and stream processing. They next go through Apache Spark and its architecture, which is built on a distributed computing approach that makes use of in-memory processing and data partitioning to achieve great speed and scalability.

So basically these articles provide a thorough introduction to the different big data approaches and technologies, as well as a discussion of the advantages and disadvantages of using Apache Spark. They can be useful for academics and practitioners looking to understand the state of the art in big data processing and select the appropriate technology for their requirements.

# CHAPTER 3 : SYSTEM DEVELOPMENT

The development of a datagen pipeline for a company that delivers the required data is not enough. Efficiency is also very important. Efficiency in terms of space and time holds a very important value. Therefore it becomes a necessity to go through various frameworks, libraries, etc before the development of the actual pipeline. Here is a detailed overview of some of such terms.

## 3.1 Related Terms

### 3.1.1 Python

**Fig 1:Python Programming Language**

Python is a high-level, interpretable programming language widely used in various fields such as web development, data analysis, machine learning, scientific computation, etc. It was originally introduced by Guido van Rossum in 1991, and since then it has become one of the most popular programming languages in the world. Python is known for its ease of use, readability, and simplicity. Its syntax is designed to be clear and concise, making it ideal for novice and experienced programmers alike. Python is also an interpreted language, which means the code can be executed without compilation, which helps speed up the development process. Another characteristic that distinguishes Python from other programming languages is it being a dynamically typed language. Variables do not need to be defined with a specified data type, and their type is decided at runtime based on the value provided to them. This increases flexibility and simplicity of use since developers may quickly assign and reassign values without worrying about type definitions. Python's popularity has risen in recent years, thanks in part to the advent of data science and machine learning. Python has been a favourite among data scientists due to the availability of sophisticated libraries such as NumPy, pandas, and scikit-learn, who appreciate its simplicity of use and robust data processing capabilities.

Features of Python are as follows:-

Python has become extremely popular in recent years. It is well-known for its straightforward syntax, ease of usage, and strong capabilities, making it an excellent choice for developers, data scientists, and amateurs alike. Some features of Python that make it such a versatile and powerful programming language are as follows:-

1. Simple and easy-to-read syntax: One of Python's most distinguishing qualities is its simple and easy-to-read syntax. Unlike many other programming languages, Python employs indentation rather than curly brackets or other symbols to express block organisation. This makes the language easier for newcomers to learn and comprehend, and it also helps to decrease the amount of code necessary to do a particular task.

2. Dynamically-typed language: Python is a dynamically-typed language, which means that variables in Python are not allocated a data type at declaration but are decided at runtime based on the value provided to the variable. This provides increased flexibility and simplicity of use since

developers may quickly assign and reassign values without having to worry about type definitions.

3. Python is an interpreted language, which means that the source code is directly run without the need for a separate compilation process. This makes it easier for developers to write and test code since they can see the consequences of their changes without having to wait for the code to compile.

4. Object-oriented programming (OOP): Python supports object-oriented programming, allowing developers to write code that is reusable and modular. OOP enables the generation of objects (class instances) and offers a framework for encapsulating data and behaviour.

5. Cross-platform compatibility: Because Python is a cross-platform language, code written on one platform may be run on another without change. As a result, it is an excellent choice for developers who need to construct programmes that can operate on numerous systems.

Python also has a huge and active community of third-party built-in libraries and programming. These libraries and frameworks make it simple for developers to obtain the resources they want to execute projects including web development, data analytics, machine learning, or anything else.

Applications and Use Cases of Python are as follows:-

Python is a popular high-level programming language that has spread to a wide range of industries and applications. It is popular among developers, data scientists, and academics because of its simple syntax, ease of usage, and huge library of modules and packages. Here are some applications where this language finds its use:-

Web development: Python's web frameworks, such as Django, Flask, and Pyramid, make it an excellent choice for web development. These frameworks offer a variety of functionality, ranging from simple web page templates to full-stack solutions with built-in database support and security measures. Python is also frequently used in tandem with front-end technologies, such as HTML, CSS, and JavaScript to create dynamic and interactive web applications.

Data Science and Machine Learning: Python is a popular choice for data scientists and machine learning practitioners because of its flexibility and strong libraries such as NumPy, Pandas, and Scikit-learn. Python enables efficient data processing and analysis, allowing users to deal with enormous datasets in a short period of time.

Scientific Computing and Numerical Analysis: Python's ability to work with arrays and matrices, together with libraries like SciPy and NumPy, makes it a good choice for scientific computing and numerical analysis. These libraries offer a variety of functions, such as linear algebra, optimisation, and statistical analysis. Python is also widely used to simulate and analyse complicated systems in scientific domains such as physics, chemistry, and biology.

Scripting and Automation: Python is an excellent choice for scripting and automation due to its simple syntax and ability to automate repetitive activities. File management, data processing, and system administration may all be automated using Python scripts. Python is also often used in DevOps, where it is used to automate deployment and monitoring of softwares.

Python libraries and frameworks in python are as follows:-

Python is a very popular programming language, and one of the reasons for this is the large number of libraries and frameworks that are available for it.

These libraries and frameworks provide strong tools for developers to make their work more efficient and productive. Python includes a number of standard libraries as part of the language itself. These libraries cover a wide range of subjects, from fundamental data types and structures to sophisticated capabilities like network programming and regular expressions.

Here are a few examples of standard Python libraries:

1. math: Provides basic mathematical functions like sin, cos, tan, and logarithms.
2. os: Provides access to operating system functionality like file operations, directory operations, and process management.
3. datetime: Provides classes for working with dates and times.
4. random: Provides functions for generating random numbers.
5. urllib: Provides functions for making HTTP requests.

Third-party libraries and frameworks for Python

While the standard libraries provide a solid foundation for Python development, most Python developers rely heavily on third-party libraries and frameworks to build their applications. There are thousands of third-party Python libraries and frameworks available, covering almost every possible use case.

Here are a few examples of popular third-party Python libraries and frameworks:

1. Flask: A lightweight web framework for building RESTful APIs and web applications.
2. Django: A full-stack web framework for building complex web applications quickly and easily.

3. NumPy: A library for working with arrays and numerical data in Python.
4. Pandas: A library for data analysis and manipulation.
5. Matplotlib: A library for creating data visualizations like charts and graphs.
6. TensorFlow: A library for machine learning and deep learning.
7. Scikit-learn: A library for machine learning algorithms like classification, regression, and clustering.
8. Pygame: A library for building games and multimedia applications.

Popular Python libraries for data science and machine learning

Python has become one of the most popular programming languages for data science and machine learning, and there are many powerful libraries available specifically for these use cases.

Here are a few examples of popular Python libraries for data science and machine learning:

1. NumPy: A library for working with arrays and numerical data in Python.
2. Pandas: A library for data analysis and manipulation.
3. Matplotlib: A library for creating data visualizations like charts and graphs.
4. SciPy: A library for scientific computing and advanced mathematics.
5. Scikit-learn: A library for machine learning algorithms like classification, regression, and clustering.
6. TensorFlow: A library for machine learning and deep learning.
7. Keras: A library for building neural networks and deep learning models.
8. PyTorch: A library for building and training neural networks.

Some Python Advanced Features and Concepts are:-

Python is a versatile and powerful programming language that has gained popularity in recent years due to its ease of use, simplicity, and broad range of applications. While Python's fundamental syntax is straightforward, it also has a number of advanced features and concepts that may be used to build complex applications and systems.

Python Object-Oriented Programming (OOP)-Object-oriented programming is a programming paradigm that organises data and functionality into classes and objects to generate modular, reusable code. Python is an object-oriented programming language, which means it supports concepts like encapsulation, inheritance, and polymorphism. Classes in Python are specified using the "class" keyword, and objects are produced with the class constructor function. The "def" and "self" keywords are used to add methods and attributes to a class, respectively.

Decorators and Metaclasses in Python-Decorators are a strong Python feature that allows functions to be changed or expanded by other functions. Decorators are useful for adding functionality to existing functions, such as logging or error checking, without changing the original function code. Decorators are created using the "@" symbol followed by the name of the decorator function. Metaclasses are another strong Python feature that allow classes to be changed or expanded by other classes. Metaclasses are used to specify how class objects should behave during execution. It may be used to add new methods or attributes to classes, change the behaviour of existing classes, or create custom class formation logic.

Python Multithreading and Multiprocessing- Multithreading and multiprocessing are two wonderful Python ideas that enable you to do numerous activities concurrently. Multithreading employs threads, which are subprocesses that operate in a single process, whereas multiprocessing employs a number of activities that occur concurrently on various CPU cores. Multithreading and multiprocessing may be utilised to offer many Python functions, and speed and scalability have substantially improved.

The Python "threading" module, which provides a simple interface for creating and managing threads, can be used for multithreading. Offering comparable interfaces for configuring and managing systems, the "multiprocessing" module can be used for multiple processing. Both modules have tools for monitoring and managing concurrent activities, as well as supporting synchronization and communication between threads and processes

Python is a high-level multi-purpose programming language that has become increasingly popular in recent years. Python's simple syntax, robust library support and cross-platform portability have made it popular with developers in projects ranging from web development to scientific computing

The future of Python is bright, with improvements and new features being added to the language all the time. The recent release of Python 3.10 has significant improvements in performance, security and usability. Python is expected to remain the top choice for developers across industries as it grows in popularity. Python is well suited to a wide variety of programming challenges, from simple scripting tasks to big data analysis to machine learning projects, thanks to its simple implementation, robust library and active community
Overall, Python is a robust and scalable programming language with various benefits for developers.

### 3.1.2 Spark

Spark is a distributed computing system designed to process large amounts of data in parallel. It was developed as a research project at AMPlab at UC Berkeley in 2009, and later became an open source project in 2010.It has gained significant popularity due to its high processing speed, memory management, fault tolerance and many more such characteristics.

**Fig 2:Apache Spark Logo**

Spark is an essential tool for big-data processing because it can handle large amounts of data that would otherwise be impossible to handle with traditional tools. Spark for businesses enables businesses to process large amounts of data in near real-time, make decisions rapidly, show patterns and trends, and get valuable insights.

Spark was developed in 2009 by Matei Zaharia and his colleagues at the AMPlab at UC Berkeley. The goal was to build a distributed computing system that could handle enormous amounts of data efficiently and swiftly. Hadoop MapReduce, a then-common standard for big data processing, was initially improved by Spark. In 2010, Spark was made accessible as an open-source project, and the big data processing sector adopted it quickly. Due to its rapid execution times, efficient memory management, and fault tolerance, Spark has been a well-liked alternative to Hadoop MapReduce. Since then, Spark has grown and altered as new features and functionalities have been added.

The importance of Spark in the realm of Big Data processing is unparalleled. Capable of parallel processing of huge volumes of data, Spark is essential to companies hoping to analyze the increasingly vast quantities of information they receives. Its support for a wide range of data sources such as HDFS, Apache Cassandra, Apache HBase, and Amazon S3 allows Spark to be used for analysis from multiple streams as well. Beyond its unrivaled processing power, Spark’s astute memory management and fault tolerance capabilities make it a truly powerful tool for Big Data applications. Spark stores data in memory for quicker analysis, compared to disk-based systems, and its fault tolerance capabilities ensure that data is neither lost nor damaged in the event of a node failure. Indeed, Spark's ability to make incredibly complex data analysis faster and more accurate, is truly invaluable.

Spark is a distributed computing system designed to process large amounts of data in parallel. The Spark infrastructure is built on a cluster computing framework, which allows data processing tasks to be distributed across clusters of computers The Spark infrastructure is flexible, allowing users to process data separately if depending on the data type and resource requirements. The Spark framework is based on a master-worker architecture, where a master node controls a group of worker nodes that perform data processing tasks. The master node controls the distribution of tasks among the worker nodes, ensuring that the tasks are executed in an efficient and fault-tolerant manner.

Spark consists of many components that work together to efficiently manage data. These factors include:

1. Spark Core: This is the core component of Spark and provides the core functionality of distributed data processing. Spark Core provides support for distributed workflows, fault tolerance, and data parallelism.

2. Spark SQL: This product provides support for structured and semi-structured data operations. Spark SQL allows users to query data with SQL-like syntax and supports data sources such as Hive, Parquet, and JSON.

3. Spark Streaming: This feature provides support for managing real-time databases. Spark Streaming enables users to process data streams in near real-time, enabling real-time decision making and monitoring.

4. Spark MLlib: This product provides support for machine learning and data mining. Spark MLlib contains various machine learning algorithms and tools for data preprocessing and feature extraction.

5. Spark GraphX: This product provides support for graph processing. Spark GraphX enables users to efficiently manipulate and analyze large graphs.

Spark can be executed in various execution modes, which  includes modes like standalone, YARN, and Mesos. Standalone mode is the default mode and is suitable for small to medium-sized clusters. YARN and Mesos are cluster managers that provide support for managing large clusters.

Spark provides support for different data abstraction and processing models, including RDDs, DataFrames, and Datasets. DataFrames provide a higher-level API for processing structured and semi-structured data. Datasets provide a type-safe API for processing data and support both structured and unstructured data. Spark is a powerful distributed computing framework that allows users to process large-scale datasets in parallel. Spark provides several data abstractions and processing models that enable users to process data in a distributed and fault-tolerant manner. In this article, we will explore some of the key data abstractions and processing models in Spark.

Resilient Distributed Datasets (RDDs) are the basic building block of Spark. RDDs are immutable distributed collections of objects that can be processed in parallel across a cluster of computers. RDDs are fault-tolerant, meaning that if a node in the cluster fails, the RDD can be reconstructed from the data on other nodes. RDDs provide two types of operations: transformations and actions. Transformations create a new RDD from an existing RDD, while actions perform some computation on the RDD and return a result to the driver program or save the result to external storage. Some examples of transformations in RDDs include map(), filter(), and flatMap(), while some examples of actions include count(), collect(), and saveAsTextFile().

DataFrames and Datasets are higher-level abstractions in Spark that provide a more structured way to work with data. DataFrames are distributed collections of data organized into named columns, similar to a table in a relational database. Datasets provide type-safe, object-oriented programming interfaces for working with structured and unstructured data. DataFrames and Datasets provide a rich set of operations, including filtering, aggregation, sorting, and joining. They can be created from a variety of data sources, including CSV, JSON, Parquet, and Hive tables. DataFrames and Datasets can also be converted to RDDs for lower-level processing if necessary.

Spark provides a wide range of transformations and actions that can be used to process data in RDDs, DataFrames, and Datasets. Basically Transformations are operations that create a new RDD, DataFrame, or Dataset from an existing one, while actions are operations that return a result to the driver program or save the result to external storage. A few examples of Transformations in spark are as follows map(), filter(), flatMap(), groupByKey(), reduceByKey(), join(), and cogroup(). A few examples of Actions in spark are as follows count(), collect(), reduce(), saveAsTextFile(), and foreach().

Caching and persistence are essential features of Spark that allow users to reuse RDDs, DataFrames, and Datasets across multiple computations. Caching an RDD, DataFrame, or Dataset stores the data in memory or on disk, reducing the need to recompute the data each time it is used in a computation. Spark provides several levels of persistence, including MEMORY_ONLY, MEMORY_ONLY_SER, MEMORY_AND_DISK, MEMORY_AND_DISK_SER, and DISK_ONLY. Choosing the right persistence level depends on the size of the data, the available memory, and the frequency of use.

Spark is a popular distributed computing framework used for big data processing. It provides a range of programming languages and APIs for working with data in distributed environments. Spark provides support for several programming languages, including Scala, Java, Python, and R. Scala is the native language of Spark, and many of its core libraries are written in Scala. Java is another popular language used with Spark and provides a more verbose syntax for working with Spark. Basically Python and R are popular languages used for data science, and their support in Spark makes it easier to use Spark for data science tasks.

Basically Spark provides several APIs for working with data, including the RDD (Resilient Distributed Datasets), DataFrame, and Dataset APIs. RDD is the basic building block of Spark and provides low-level APIs for working with distributed data. DataFrame and Dataset APIs provide a more structured and type-safe way to work with data, similar to working with tables in a relational database.

The DataFrame and Dataset APIs are similar in structure, but Dataset provides stronger type-safety guarantees than DataFrame. Basically both APIs provide a wide range of operations for working with data, including filtering, aggregation, and joining.

Spark SQL is a Spark module that allows you to deal with structured data using a programming interface. Spark SQL allows users to query data using SQL-like syntax, making data interaction easier for persons with SQL knowledge. Spark SQL can read and write data in a variety of formats, including JSON, CSV, and Parquet. There are certain advantages to utilising Spark SQL over a traditional SQL engine. To begin, it effortlessly connects with Spark's existing APIs, allowing customers to mix and match several APIs for data processing. When dealing with huge data sets, Spark SQL offers data aggregation and durability, which may significantly boost throughput. Finally, because Spark SQL offers batch and streaming data processing, it is an excellent choice for real-time data processing.

Spark Streaming is a Spark module that supports real-time data processing. Users may manage data streams in real time using the same API that is used for batch data processing with Spark Streaming. Spark Streaming supports a variety of data sources, including Kafka, Flume, and HDFS.Fault tolerance, user scalability, and support for window-based workflows are just a few of the features that make Spark Streaming excellent for real-time applications. Spark Streaming also integrates with Spark's other APIs, allowing you to easily combine real-time and batch processing.

Spark is a popular distributed computing platform for processing massive data that is meant to run on machine clusters. The Spark cluster is made up of numerous nodes, each of which is a machine in the cluster. A Spark cluster consists of two types of nodes: master nodes and worker nodes.So basically, the master node maintains the Spark application and supervises the worker nodes operations. The worker nodes are in charge of the duties given to them by the master node.

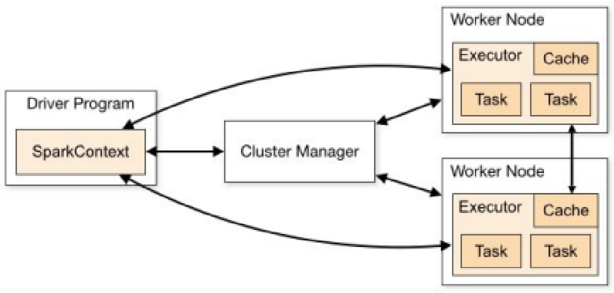


**Fig 3:Spark Cluster model**

Spark can be deployed to a cluster using one of three different cluster managers: Standalone, YARN, and Mesos.

Standalone mode is the easiest way to deploy Spark in a cluster. So basically, In this mode, Spark is its own cluster manager, and the user can install Spark through a simple command-line interface. YARN is another popular cluster manager used in Spark. YARN is a part of Hadoop and is widely used in Hadoop clusters. Spark can be run on YARN using a simple configuration file. Mesos is another popular cluster manager used in Spark. Mesos provides a distributed system kernel that pulls in CPU, memory, storage, and other compute resources to create a shared pool of resources that Spark can use.

Managing Spark clusters can be challenging, and there are several tools available to help manage Spark clusters. Some of the most popular Spark cluster management tools which can be used to manage spark clusters include Apache Ambari, Cloudera Manager, and Hortonworks Data Platform.Basically, Apache Ambari is an open-source management tool used to manage Hadoop and other related technologies, including Spark. Ambari provides an easy-to-use web-based interface for deploying, configuring, and monitoring Spark clusters. Cloudera Manager is a proprietary management tool used to manage Hadoop and related technologies, including Spark.Basically Cloudera Manager provides an easy-to-use web-based interface for deploying, configuring, and monitoring Spark clusters. Hortonworks Data Platform (HDP) is another popular management tool used to manage Hadoop and related technologies, including Spark. HDP provides an integrated solution for managing Spark clusters and includes features such as automated provisioning, monitoring, and troubleshooting.

Spark is a powerful distributed computing framework that provides a wide range of data processing capabilities, including machine learning and graph processing. A small overview of  the machine learning and graph processing capabilities of Spark, and how they can be used for big data processing are as follows:

Machine Learning in Spark: Spark includes a strong machine learning framework known as MLlib, which allows users to create scalable machine learning models. Machine learning methods supported by MLlib include classification, regression, clustering, and collaborative filtering. MLlib is meant to operate with Spark's main API, and it includes a number of high-level APIs for developing machine learning models, such as the ML pipeline API and the DataFrame-based API.

Basically the above-mentioned ML pipeline API is a high-level API that provides a straightforward and intuitive approach to design machine learning pipelines. A pipeline is a series of processes used to turn raw data into a format that a machine learning algorithm can use. The pipeline API makes it simple to create sophisticated pipelines.By offering a collection of high-level abstractions for data transformation, feature extraction, and model training, the pipeline API makes it simple to design large machine learning pipelines. Another high-level API is the DataFrame-based API, which provides a straightforward and easy approach to create machine learning models using

Spark's DataFrame API. The DataFrame-based API contains high-level abstractions for data preparation, feature extraction, and model training.

Spark Graph Processing: GraphX, a powerful graph processing package provided by Spark, enables developers to create scalable graph algorithms. GraphX is built on top of the main Spark API, and provides enhanced APIs for developing graph algorithms. PageRank, rectangle count, associated content, and label width are just a few of the graph algorithms supported by GraphX. GraphX also provides many amazing APIs for graph applications, such as the Graph API and the Pregel API. Graph API is a high-level API that allows you to create graph algorithms in a simple and logical way. The Graph API provides high-level abstractions for developing graph algorithms, such as vertex and edge RDDs for defining the vertices and edges of a graph. Pregel API is another high-level API that makes it easy to build graph algorithms using the Pregel programming model. The pregel programming model is a message-based program that transfers messages between vertices to provide shortcuts for graph algorithms.

Spark is a powerful big data processing tool that has been widely adopted by industries of all kinds. From e-commerce to healthcare, Spark is a versatile and efficient solution for handling large datasets. The most popular use cases of Spark and real-world examples of its application are as follows:-

1. E-Commerce: E-commerce companies process a lot of data every day, from customer interactions to inventory management. Spark provides real-time solutions to process this data, enabling eCommerce businesses to gain insights into customer behavior and make data-driven decisions For example, Amazon uses Spark for its customized product recommendations for each, based on user behavior and purchase history.

2. Financial information: Financial institutions produce a wealth of information every day, from stock prices to trading history. Spark provides real-time solutions to process this data, enabling financial institutions to monitor market trends and create investment decisions such as Capital One using Spark to detect fraud, detect transactions as it creates illusion and prevents fraudulent activities.

3. Health care: The healthcare industry processes a lot of data every day, from patient records to medical images. Spark provides real-time solutions to process this data, empowering healthcare providers to improve patient outcomes and reduce costs. For example, the University of California, San Francisco uses Spark to analyze medical imaging data, identify patterns, and predict disease outcomes.

4. Telecommunications: Telecom companies generate a lot of data every day, from call logs to networks. Spark provides real-time solutions to process this data, enabling telecom companies to monitor network performance and identify potential issues For example, AT&T uses Spark for its predictive maintenance program.

Real-World Examples of Spark Usage are as follows

1. Uber: Uber is a transportation network company that uses Spark for its data processing needs. Uber uses Spark to analyze trip data and optimize its pricing strategy, ensuring a good match between riders and drivers. Spark also helps Uber forecast demand in real time, allowing the company to better allocate drivers.

2. Netflix: Netflix is a popular streaming service that uses Spark for its data processing needs. Netflix uses Spark to optimize its content recommendations, ensuring that users are shown relevant content based on their viewing history.

Spark also helps Netflix manage network performance, ensuring users can stream content with ease.

3. IBM Watson: IBM Watson is a cognitive computing platform that uses Spark for its data processing needs. IBM Watson uses Spark to process vast amounts of unstructured data, enabling it to understand natural language and generate insights. Spark also helps IBM Watson to identify patterns in data, making it a powerful tool for predictive analytics.

In summary, Spark is a fast and efficient framework for running distributed computing on large datasets. It provides flexible configuration models and APIs that can be used to develop complex data processing workflows. Spark's support for multiple programming languages and compatibility with various cluster computing frameworks also make it a versatile option for big data processing

Looking ahead, Spark's future prospects are very promising. With the increasing demand for real-time data processing and machine learning, Spark is expected to become an increasingly important tool for dealing with big data as new use cases and applications emerge, making Spark available improve and improve, and further solidify its position as a leading big data provider management system

In conclusion, Spark provides a powerful and flexible solution to handle large data processing applications. Its ease of use, scalability, and wide range of features and benefits make it a valuable tool for organizations of all sizes and industries. As the demand for big data processing increases, Spark will undoubtedly play a key role in shaping the future of data-driven decision-making.

### 3.1.3 Pyspark

Big data has become an essential component of modern enterprises and businesses. As data evolves, methods and technology that can efficiently manage and analyse massive volumes of data are required. PySpark is one such technology that has grown in popularity in recent years. PySpark is a Python API for Apache Spark, a powerful distributed computing framework designed for big data processing. This allows developers to write Spark applications using the Python programming language, which is one of the most popular languages in the data science and machine learning communities

PySpark provides a flexible interface for Spark programming and Python. It enables developers to leverage the power of Apache Spark for big data processing tasks, but retain the simplicity and flexibility of Python PySpark offers a superior API for distributed data processing and machine learning, making it an excellent choice for data scientists and developers who want to process more data efficiently

PySpark has become an essential tool for big data processing for several reasons. First, Python is a popular programming language among data scientists and developers. This popularity has led to the creation of a vast ecosystem of Python libraries for data science, machine learning and artificial intelligence, such as NumPy, Pandas, and Scikit-learn. Second, Apache Spark, the underlying distributed computing framework, is designed to handle large amounts of data in a distributed and parallel manner, making it ideally suited for big data processing Spark uses an API powerful variety for distributed data processing, including data processing, streaming, and machine learning. Finally, PySpark makes it easy for developers to use the power of Spark from Python, a language that many data scientists and developers are already familiar with This makes it easy for developers to get started with Spark and benefit from its power if distributed electronically.

One of the fundamental concepts of PySpark is RDDs, or Resilient Distributed Datasets. RDDs are a distributed collection of objects that can be processed in parallel across a cluster of computers. RDDs can be created from data stored in files, databases, or other sources, and they can be transformed and processed using PySpark's APIs.

PySpark provides a set of APIs for transforming RDDs. Transformations are operations that produce a new RDD from an existing one. Examples of transformations include map, filter, and reduce. Actions, on the other hand, are operations that return a value to the driver program or write data to an external storage system. Examples of actions include count, collect, and save.

PySpark uses lazy evaluation, which means that transformations on RDDs are not executed immediately. Instead, PySpark builds a directed acyclic graph (DAG) of transformations and actions and only executes them when an action is called. This approach is more efficient than immediately evaluating every transformation, as it allows PySpark to optimize the execution plan and avoid unnecessary computations.

PySpark is a well-known distributed computing technology that enables users to analyse massive volumes of data in an efficient and scalable way. PySpark provides a number of libraries and components in addition to its core foundations like as RDDs, transformations, and actions, making it a formidable tool for data scientists and engineers.

Some Pyspark Libraries are as follows:-

- Spark SQL, one of the most popular PySpark libraries, provides a high-level interface for working with structured and semi-structured data. Users may simply conduct SQL queries on huge datasets using Spark SQL, as well as perform complex analytics with built-in functions and operators.

- Spark Streaming, another major component of PySpark, allows users to handle real-time data streams in parallel over a cluster of computers. Users may utilise Spark Streaming to create real-time data pipelines that analyse data as it is created, providing rapid and accurate insights into streaming data.

- Another key feature of PySpark is MLlib, or the Machine Learning library, which contains a variety of machine learning techniques and data analysis tools. Users can use MLlib to build and train models on large data sets, as well as perform sophisticated analyzes such as clustering and classification.

- Finally, GraphX is a graph application that allows users to perform powerful graph calculations on large data sets. The well-designed GraphX API allows users to quickly define and modify graphs, create graph algorithms, and create new graph algorithms with GraphX.

Overall, PySpark offers a wide range of libraries and features, making it an effective tool for data scientists and engineers. whether they need to handle real-time data streams, build and train machine learning models, or perform complex calculations. PySpark is a strong tool for dealing with massive data, but it may be challenging to write efficient code and maintain applications. There are various best practises for PySpark development like including efficient code, debugging approaches, and development tools that one should follow.

Writing high-performance code that can handle massive volumes of data is one of the most difficult aspects of PySpark development. Users must first understand the various RDD variables and actions available in PySpark and how you can use them to optimise Pyspark code. For example, transformations such as map, filter, reduceByKey, and so on can frequently be used to conduct operations on RDDs. Furthermore, methods such as caching and partitioning may be used to boost the speed of PySpark code.

Because Spark is dispersed, developing PySpark apps might be difficult. Using logging statements to get information about the progress and status of your application is a valuable technique of troubleshooting PySpark applications. You may also analyse your application's behaviour using PySpark's built-in debugging tools, such as the Spark UI. Another good debugging strategy for PySpark apps is to run your code locally initially with tiny datasets before scaling up to bigger datasets on the Spark cluster. This might assist you in swiftly identifying and correcting mistakes.

Many PySpark development tools are available to assist simplify and speed the development process. PyCharm, for example, is a popular Python IDE that incorporates PySpark functionality, such as syntax highlighting, code completion, and debugging. Jupyter Notebook and Apache Zeppelin, for example, provide an interactive environment for working with PySpark, letting users to test and iterate your code fast. PySpark Use Cases: Big Data Processing, Machine Learning, and Real-Time Streaming Analytics. Python API for Apache Spark PySpark is a powerful tool for processing and analysing large amounts of data. It provides high-level connectivity for distributed computing and helps data scientists and developers to construct complicated data processing applications using Python. Beyond massive data processing, PySpark's capabilities include machine learning and real-time flow analysis. In this essay, we will look into the PySpark deployment environment in various scenarios.

Big Data processing: PySpark is a popular tool for big data processing. Using distributed computing approaches, it can efficiently handle big data processing processes. PySpark is a flexible tool for large data processing since it can handle organised, semi-structured, and unstructured data.

1. Extract, Transform, and Load (ETL) PySpark can take data from numerous sources, transform it to an arbitrary format, and load it into the target system.

2. Data cleaning and preprocessing: PySpark may be used to clean data, validate data, normalise data, and transform data. This is especially helpful when dealing with unstructured data, which needs substantial cleaning and upkeep.

3. Data analysis: PySpark can execute complicated data analysis jobs on large amounts of data, such as making statistical choices, creating reports, and discovering patterns and trends.

Machine learning: PySpark also provides a rich set of machine learning libraries and tools that can be used for a variety of applications. PySpark's machine learning libraries are built on top of Spark's distributed computing framework, enabling data scientists to efficiently execute large machine learning projects Here are some of PiSpark's most popular machine learning use cases:

1. Predictive modeling: PySpark can be used to build predictive models that can be used for various applications such as fraud detection, customer segmentation, recommendation systems and more

2. Grouping and classification: PySpark can be used to perform clustering and classification tasks such as combining similar objects and dividing objects into predefined groups.

3. Natural Language Processing (NLP): PySpark can be used for NLP tasks such as sentiment analysis, text segmentation, and entity recognition.

Real-time streaming analysis: PySpark can also be used for real-time streaming analysis. It provides a streaming API that can be used to process data streams in real time. PySpark's streaming capabilities make it a popular choice for applications such as:

1. Fraud Detection: PySpark can be used to detect fraudulent transactions in real time by analyzing transaction databases.

2. Social Media Monitoring: PySpark can be used to monitor social media feeds in real time and analyze trends and sentiment.

3. IoT data processing: PySpark can be used and analyzed data streams from IoT devices such as sensors and wearables.

PySpark is a robust open source distributed computing platform that has transformed huge data processing, analysis, and administration. Spark SQL, Spark Streaming, MLlib, and GraphX are among the PySpark libraries and functionalities that have been tested. PySpark has garnered great popularity in the large data processing and machine learning fields because to its scalability, adaptability, and ease of use. It contains sophisticated features and tools that let users to handle and analyse enormous volumes of data with ease. PySpark's machine learning and real-time streaming analytics features make it a formidable tool for corporations and organisations across industries.

An important advantage of PySpark is its integration with the Python programming language, which has a large developer community and a wide range of third-party libraries and programs This integration enables data scientists and developers to use Python libraries simple, flexible and exploits the rich ecosystem. To get the most out of PySpark, developers and data scientists should follow best practices, including writing efficient code, maintaining applications, and deploying development tools to optimize their workflow. These actions can help improve the performance and reliability of PySpark applications and make it easier to perform maintenance and upgrades. PySpark is projected to continue to evolve and improve its capabilities in the future. PySpark may become an increasingly significant tool for businesses and organisations that need to effectively handle and analyse massive volumes of data as big data grows and demand for real-time analytics grows.

In conclusion, PySpark is a strong tool for massive data processing, machine learning, and real-time leak detection. Its adaptability, scalability, and simplicity of use make it a great tool for enterprises and organisations of all sizes. Developers and data scientists may use PySpark's capabilities and achieve their data processing and analytics goals by employing development tools that adhere to best practises.

### 3.1.4 Spark Clusters

Apache Spark is a distributed computing system designed to process large amounts of data simultaneously across a set of computers. A Spark cluster is a group of machines that work together to process data, and allows Spark users to distribute data and workloads across this group to improve performance and scalability In this article we will explore the concept of clusters in Spark terms, how they work, and their benefits. A Spark cluster is a group of machines that work together to process data in parallel. Each machine in the cluster is called a node and can be either a master node or a worker node. The master node distributes tasks among worker nodes and organizes the entire workflow. The worker nodes manage the data and send the results back to the master node.

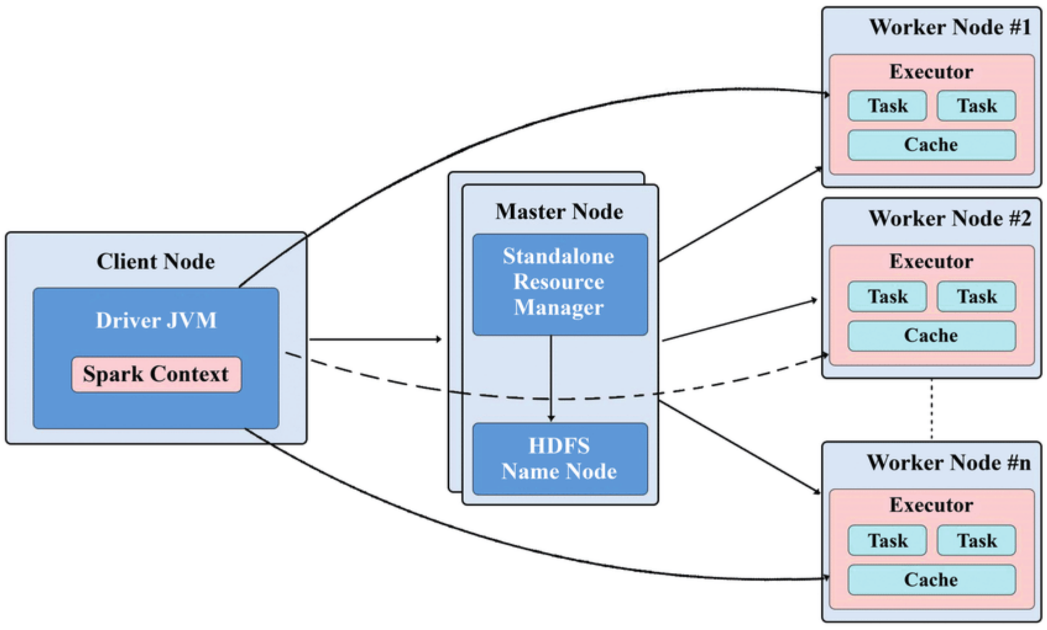


**Fig 4:Spark Cluster Architecture**

Benefits of Using a Spark Cluster are as follows:-

Using a Spark cluster provides several benefits for processing large volumes of data:

1. Scalability: A Spark cluster allows users to scale processing power up or down as needed by adding or removing nodes. This makes it easy to handle large volumes of data and meet changing business needs.

2. Speed: Processing data in parallel across a cluster of machines can significantly reduce the time it takes to analyze large volumes of data.

3. Fault tolerance: Spark is designed to be fault-tolerant, meaning that if a node fails during processing, the work can be automatically redistributed to other nodes in the cluster.

4. Cost-effectiveness: Using a Spark cluster can be more cost-effective than using a single high-powered machine, as it allows users to use a group of lower-cost machines to process data in parallel.

The Spark cluster distributes data and tasks across numerous processors in simultaneously. The master node keeps track of the whole cluster and manages workload allocation among worker nodes. The worker node performs simultaneous tasks and reports the results to the master node. Spark is intended to manage data allocation and workloads on clusters automatically, ensuring that each node works on the portion of the data set that it is most equipped to handle.

While employing a Spark cluster has numerous benefits, there are also drawbacks to consider.

1. Setup and configuration of a Spark cluster may be complicated and time-consuming, requiring knowledge in distributed computing and networking.

2. Maintenance: Maintaining a Spark cluster necessitates regular monitoring and monitoring to verify that all nodes and the cluster are operational.

3. Data locality: When data is not stored locally on the same system that is processing, Spark may face performance difficulties. In the nineteenth century, this necessitates careful consideration of data storage and dissemination in a cluster.

Spark Cluster is a powerful tool for processing large amounts of data in parallel, providing scalability, speed, fault tolerance and cost savings. While there are challenges to consider, the benefits of using a Spark cluster make it a valuable tool for many organizations that want to process big data. By understanding how Spark clusters work and carefully considering their design and maintenance, organizations can take full advantage of the benefits it offers.

### 3.1.5 Airflow

Apache Airflow is an open-source platform that allows developers and data scientists to systematically write, schedule, and track workflows, which consist of various tasks to be performed in a specific order. Airflow was developed at Airbnb in 2015. Since meanwhile, data operations penetrate organizations of all sizes It has become a popular management tool. In this article we will explore the features and benefits of Airflow and provide a guide to getting started with the platform.

**Fig 5: Apache Airflow Logo**

Apache Airflow Features:

1. Workflow management: Apache Airflow can run workflows through Python code or through its user interface, making it easy to create, configure, and manage complex workflows You can create objects based between services,

edit them warning, and create custom plugins to extend its functionality and you can.

2. Dynamic workflows: Dynamic workflows allow you to adjust your workflows in real time as your data changes. This makes it easier to adapt to changing business needs and ensures that your business processes stay up-to-date.

3. Scalability: Airflow is designed to be scalable, allowing you to manage workflows across multiple devices and clusters. It also supports distributed execution, and allows you to execute tasks in parallel on different machines.

4. Expandability: Airflow has a modular architecture that makes it easy to expand its functionality. You can create custom operators and sensors, integrate with external systems, and even create your own plugins.

5. Visibility and Monitoring: Airflow provides elegant tools for performance monitoring and troubleshooting. You can view logs and metadata for individual tasks, track business progress, and set up alerts and reports to ensure your workflow is running smoothly.

Advantages of Apache Airflow:

1. Increased Efficiency: Airflow helps you to automate repeated operations, saving you time and effort while managing complicated workflows. This can lead to increased team efficiency and production.

2. Increased Flexibility: Because Airflow is dynamic, it is simpler to react to changing business requirements and data environments. You may change your processes in real time to keep them up to date and relevant.

3. Error Reduction: Airflow may assist decrease the risk of mistakes and guarantee that your workflows are completed correctly and consistently by automating operations and eliminating the need for manual intervention.

4. Improved Collaboration: Airflow provides a centralised platform for managing processes, allowing teams to interact and share resources more easily. You may delegate work to other team members, monitor progress, and give comments and assistance.

5. Cost Savings: Airflow can assist minimise the expenses associated with managing complicated workflows by automating operations and enhancing efficiency. It can also assist to lessen the need for manual intervention, lowering the chance of mistakes and downtime.

Getting Started with Apache Airflow:

1. Install Airflow: Airflow can be installed using pip, the Python package manager. You'll also need to install any necessary dependencies and libraries.

2. Set up a Database: Airflow requires a database to store metadata and information about your workflows. You can use a variety of databases, including PostgreSQL, MySQL, and SQLite.

3. Configure Airflow: Once you've installed Airflow and set up your database, you'll need to configure Airflow's settings. This includes setting up authentication, specifying the location of your DAGs (Directed Acyclic Graphs), and configuring the executor that will be used to execute your tasks.

4. Create DAGs: DAGs are the core building blocks of workflows in Airflow. You can create DAGs using Python code or through the Airflow UI. Users

need to define the tasks that make up your workflow, specify their dependencies, and configure any configs.

Apache Airflow is a powerful platform for managing, scheduling, and monitoring data pipelines. It has gained popularity among data engineers and data scientists for its flexibility, scalability, and robustness. Here are some of the use cases where Apache Airflow can be helpful:

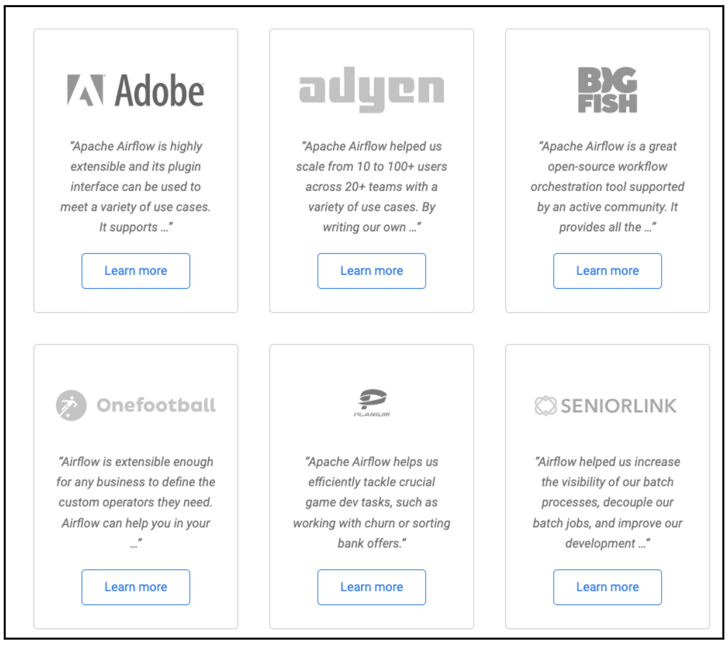
1. ETL workflows: Apache Airflow is suited for creating complicated ETL processes (Extract, Transform, Load). It has an easy-to-use interface for establishing jobs, dependencies, and scheduling, making it straightforward to manage and monitor data pipelines.

2. Machine learning: Machine learning is a multi-step process that includes data preparation, model training, and assessment. Apache Airflow can assist data scientists manage and monitor the whole workflow by automating these procedures. Airflow, for example, may schedule data preparation chores, begin model training, and monitor model performance.

3. Data processing: Apache Airflow may be utilised in a distributed setting to handle massive amounts of data. It is capable of scheduling jobs across a cluster of computers and coordinating task execution across several nodes. This can help data engineers to build data pipelines that are scalable and fault-tolerant.

6. DevOps: Apache Airflow may be used to organise and automate DevOps processes including software development, deployment, and testing. It has the ability to schedule tasks that start builds, deploy apps, and execute tests. Airflow also integrates with technologies like as Jenkins, GitHub, and Docker, making it simple to build end-to-end pipelines that automate the full DevOps process.

Finally, Apache Airflow is a flexible platform that can be utilised for a variety of use cases, ranging from ETL to machine learning to DevOps. It is a popular option among data engineers and data scientists because of its flexible design, sophisticated scheduling features, and large ecosystem of plugins. Organisations may use Apache Airflow to automate data pipelines, enhance data quality, and boost productivity.



"Apache Airflow is highly extensible and its plugin interface can be used to meet a variety of use cases. It supports ..."

"Apache Airflow helped us scale from 10 to 100+ users across 20+ teams with a variety of use cases. By writing our own ..."

"Apache Airflow is a great open-source workflow orchestration tool supported by an active community. It provides all the ..."

"Airflow is extensible enough for any business to define the custom operators they need. Airflow can help you in your ..."

"Apache Airflow helps us efficiently tackle crucial game dev tasks, such as working with churn or sorting bank offers."

"Airflow helped us increase the visibility of our batch processes, decouple our batch jobs, and improve our development ..."

**Fig 6: Real world use cases of Apache Airflow**

## 3.2 Development

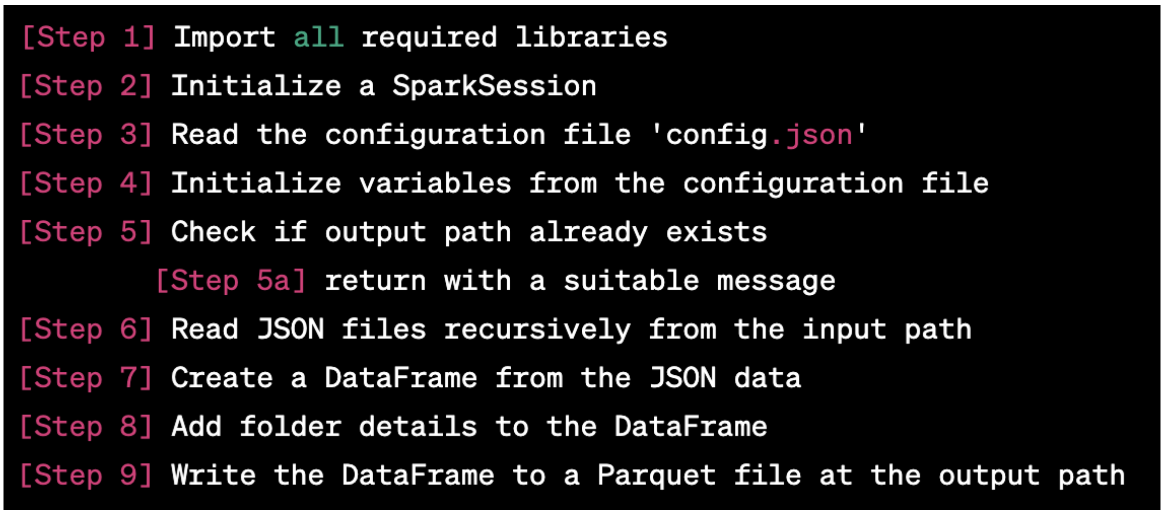
The data received is from the patients of various hospitals across the globe. The task is to reduce and preprocess this massive data so as to convert it to a form that is more space efficient and can be used directly for further analysis. Therefore we must come up with a pipeline that can do the preprocessing. It is ought to be made sure that the datagen pipeline is automated i.e., each process can be run after another automatically without intervention. Here are the details of the development process:

### 3.2.1 Raw Data

The patient data is received from hospitals in the form of JSONs. JSON is a format commonly used to exchange information. The data is divided into many folders where each folder has many JSON files. Each JSON file contains multiple single lined JSONs. One such file can be of a size nearly 400-500 MBs. Making each folder of the size of 20-30 GBs. Hence the collection of such folders was of the size 75 Tbs. The data is hence massive and requires efficient storage and processing. So the data recieved is in the form of JSONs. Each JSON can be imagined as a row in a table. Hence these JSONs combine to generate a huge table. The JSONs contain many fields which can be interpreted as a column. Each JSON can have almost 700 unique columns at maximum. However all this data isn't required and is futile. Only 100 selected columns are required for the required task. Hence processing requires removing the unnecessary columns and keeping only the required ones. This will reduce the space taken up by data considerably. Not just removing columns is necessary but also adding a few columns is required, the details shall be discussed further in the implementation.

### 3.2.2 Implementation

The implementation blueprint is designed so as to match the requirements asked. The first step would be to process each folder. Processing requires to read JSONs in each folder and first of all create a dataframe for each. This is done because of inconsistency in the data as the fields (columns) in each JSON might be different, hence a union would be done. The entries for which values are absent are filled by null values. As the whole data will be merged into one single dataset it will be essential to be able to trace back which data entry belonged to which folder, as each folder has a different set of medical data. Therefore adding folder names to each entry is necessary. Also filename is appended so that the value can be traced back if required. Therefore we see a need to process each folder separately.



```
[Step 1] Import all required libraries
[Step 2] Initialize a SparkSession
[Step 3] Read the configuration file 'config.json'
[Step 4] Initialize variables from the configuration file
[Step 5] Check if output path already exists
        [Step 5a] return with a suitable message
[Step 6] Read JSON files recursively from the input path
[Step 7] Create a DataFrame from the JSON data
[Step 8] Add folder details to the DataFrame
[Step 9] Write the DataFrame to a Parquet file at the output path
```

**Fig 7: Pseudocode for data preprocessing**

The data after going through the following process must be written to a disk. The raw data hence can be discarded and operations be performed on the data now generated. This data is clean and must occupy less space. But still the data must be stored such that it occupies less space. Hence parquet format is chosen to store the data after preprocessing.

But before discarding the data it must be ensured that the data generated is complete and error free. Hence data validation must be done before proceeding any further. This is one in the following manner: Firstly the number of lines are counted in each file of each folder and compared to the count of rows of that file in the parquet generated. The second test is a random sampling test where some random JSONs are selected from the files and are searched in the parquet. The sample size is only 0.0005% of the actual dataset. This might seem to be a small portion but it still contains almost 40,000 values and therefore searching these many values in a huge dataset is still a big task. Therefore instead of searching each value in the dataframe (generated from parquet) we perform an inner join. Inner Join seems to be a more complex task in terms of time. However we employ the characteristics of spark to make it a yet simpler task than searching each term. Here we employ broadcast join.

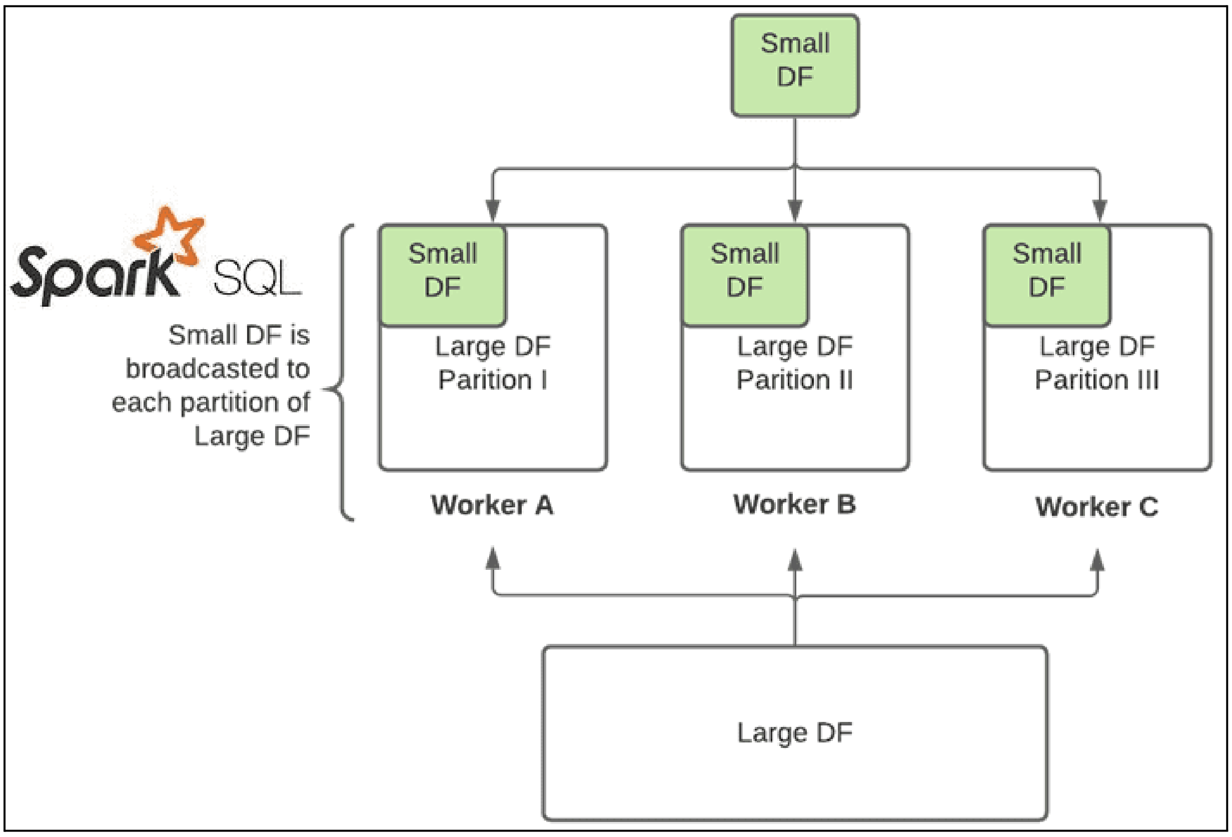
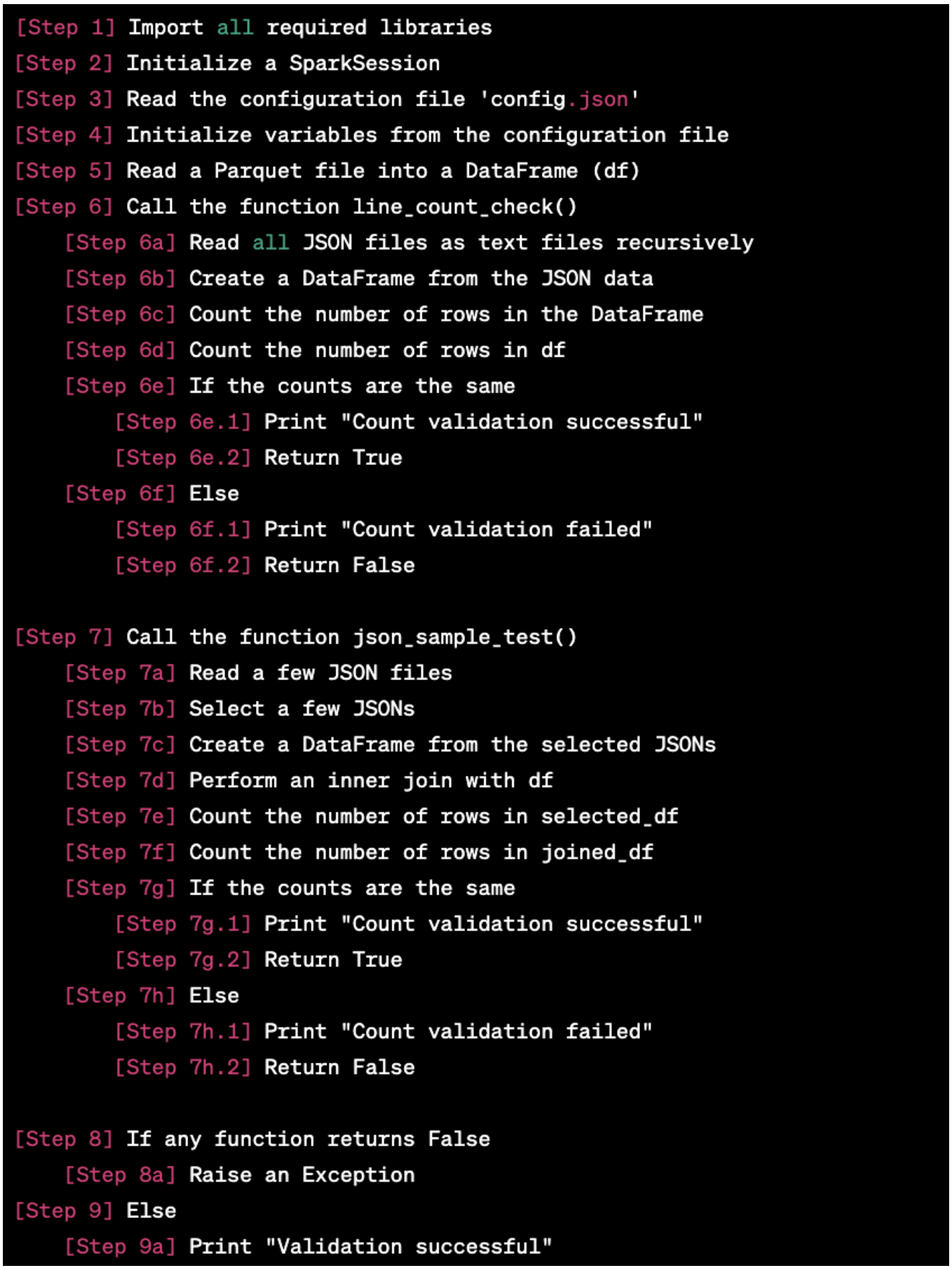


**Fig 8: Broadcast join in spark**

The broadcast join simply sends the smaller table of the two to each executor where it is joined with the same values and returned to the master node which compiles all the results and returns to the driver code. Thereby making the join process faster. Once all the values are found we are done with our data validation process. Hence this data can be now saved and the raw data can be discarded.



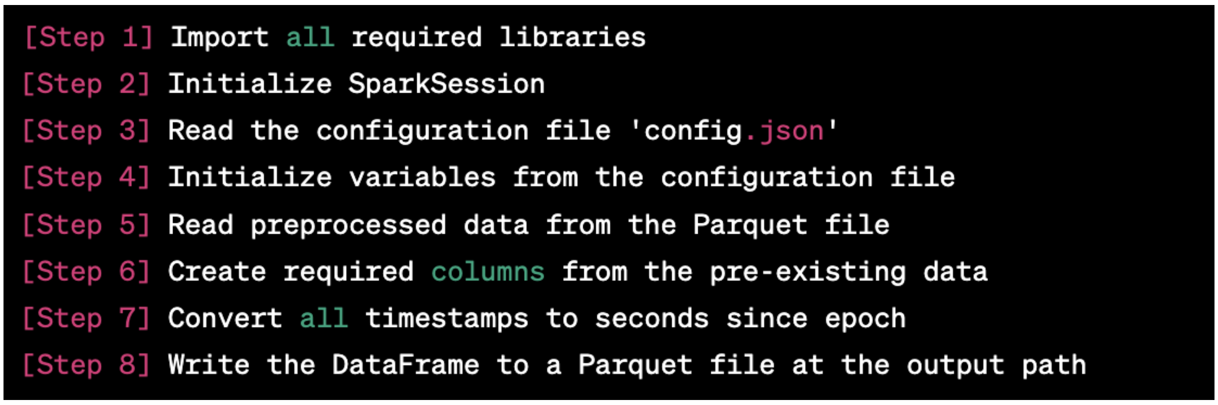
```
[Step 1] Import all required libraries
[Step 2] Initialize a SparkSession
[Step 3] Read the configuration file 'config.json'
[Step 4] Initialize variables from the configuration file
[Step 5] Read a Parquet file into a DataFrame (df)
[Step 6] Call the function line_count_check()
    [Step 6a] Read all JSON files as text files recursively
    [Step 6b] Create a DataFrame from the JSON data
    [Step 6c] Count the number of rows in the DataFrame
    [Step 6d] Count the number of rows in df
    [Step 6e] If the counts are the same
        [Step 6e.1] Print "Count validation successful"
        [Step 6e.2] Return True
    [Step 6f] Else
        [Step 6f.1] Print "Count validation failed"
        [Step 6f.2] Return False

[Step 7] Call the function json_sample_test()
    [Step 7a] Read a few JSON files
    [Step 7b] Select a few JSONs
    [Step 7c] Create a DataFrame from the selected JSONs
    [Step 7d] Perform an inner join with df
    [Step 7e] Count the number of rows in selected_df
    [Step 7f] Count the number of rows in joined_df
    [Step 7g] If the counts are the same
        [Step 7g.1] Print "Count validation successful"
        [Step 7g.2] Return True
    [Step 7h] Else
        [Step 7h.1] Print "Count validation failed"
        [Step 7h.2] Return False

[Step 8] If any function returns False
    [Step 8a] Raise an Exception
[Step 9] Else
    [Step 9a] Print "Validation successful"
```

**Fig 9: Pseudocode for data preprocessing validation**

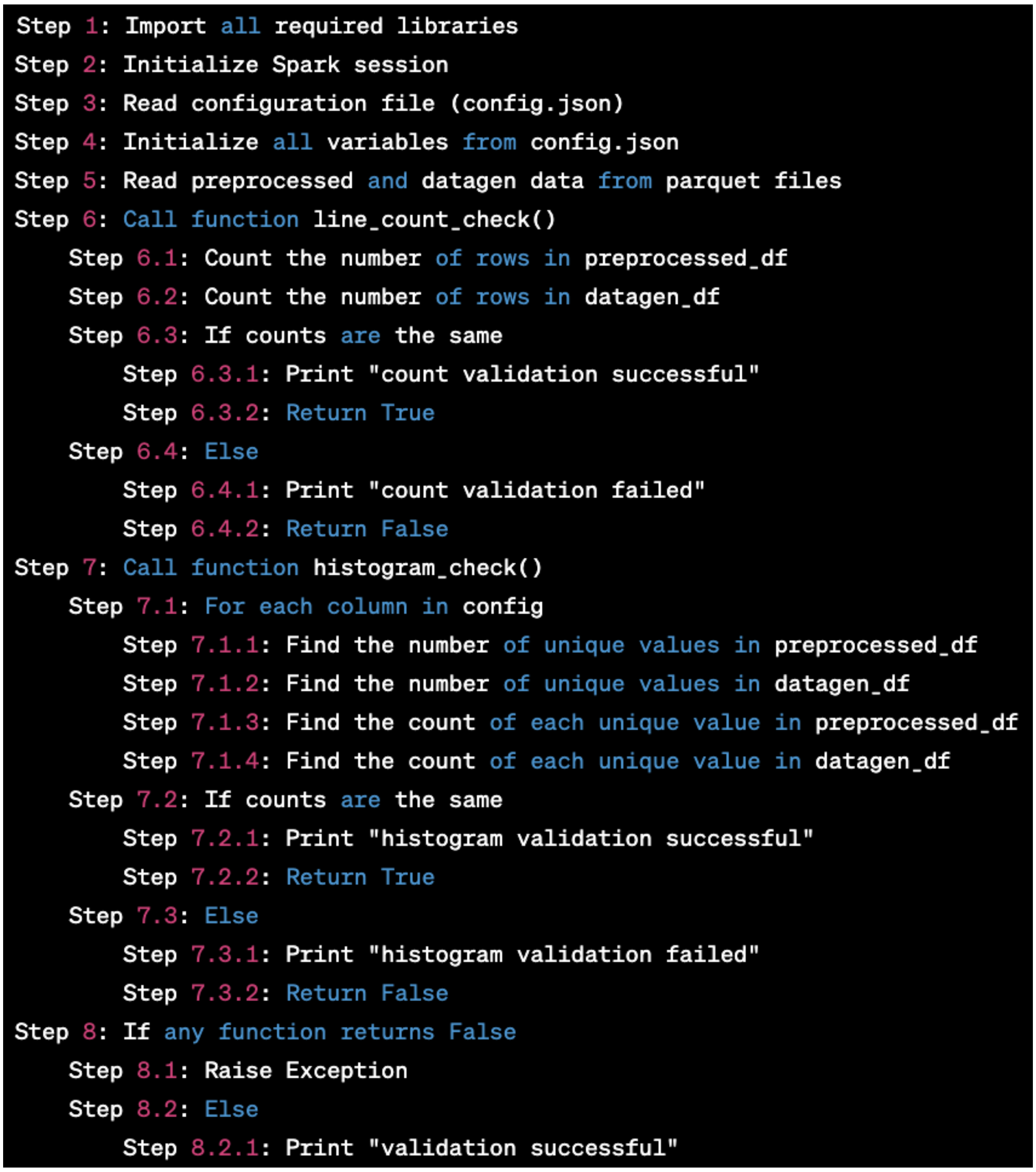
Having gone through this process the data now remaining needs to go through some more processing before it can be used for any analysis. Firstly each patient is recognized by  a string that is too long and can be reduced to a smaller string by finding a hash value. This ensures that each unique string is matched to a unique hash value which takes up far less space. Not just space but these hash values match the values present in the patient records held by the company where each patient is recognized by the hash value. This is done so that the patient cannot be traced back from his/her medical records. Thereafter we can also find the patient details for each row and append it to the data so that analysis of data can be done properly. After this minor changes can be done like changing date time given in the columns in a particular format to seconds since epoch for better and easier analysis.



```
[Step 1] Import all required libraries
[Step 2] Initialize SparkSession
[Step 3] Read the configuration file 'config.json'
[Step 4] Initialize variables from the configuration file
[Step 5] Read preprocessed data from the Parquet file
[Step 6] Create required columns from the pre-existing data
[Step 7] Convert all timestamps to seconds since epoch
[Step 8] Write the DataFrame to a Parquet file at the output path
```

**Fig 10: Pseudocode for data generation**

In the end it is again essential to verify if the contents of the dataframe so generated are correct. Again two checks will be executed, firstly we simply match the counts of the two parquet and see if they are the same or not. Secondly we will check the unique values of a few columns where we know that the number of unique values are finite or handful. If the number of unique values are the same and their frequencies are the same then we can say that the data is correct and  validation is successful.



```
Step 1: Import all required libraries
Step 2: Initialize Spark session
Step 3: Read configuration file (config.json)
Step 4: Initialize all variables from config.json
Step 5: Read preprocessed and datagen data from parquet files
Step 6: Call function line_count_check()
    Step 6.1: Count the number of rows in preprocessed_df
    Step 6.2: Count the number of rows in datagen_df
    Step 6.3: If counts are the same
        Step 6.3.1: Print "count validation successful"
        Step 6.3.2: Return True
    Step 6.4: Else
        Step 6.4.1: Print "count validation failed"
        Step 6.4.2: Return False
Step 7: Call function histogram_check()
    Step 7.1: For each column in config
        Step 7.1.1: Find the number of unique values in preprocessed_df
        Step 7.1.2: Find the number of unique values in datagen_df
        Step 7.1.3: Find the count of each unique value in preprocessed_df
        Step 7.1.4: Find the count of each unique value in datagen_df
    Step 7.2: If counts are the same
        Step 7.2.1: Print "histogram validation successful"
        Step 7.2.2: Return True
    Step 7.3: Else
        Step 7.3.1: Print "histogram validation failed"
        Step 7.3.2: Return False
Step 8: If any function returns False
    Step 8.1: Raise Exception
    Step 8.2: Else
        Step 8.2.1: Print "validation successful"
```

**Fig 11: Pseudocode for data generation validation**

All these tasks shall take up a long time hence automation is necessary. Automation means that these processes are triggered without any human intervention whenever the one preceding it succeeds. For this we use Apache Airflow. Here each process is represented by a DAG( Directed Acyclic Graph). When one dag completes the control automatically moves to the next process or node in the dag.This ensures that processes continue one after another until and unless an exception occurs. Hence no need to monitor the tasks all around the day.(As each task may take up to 4-6 hours running on a 29 node spark cluster)

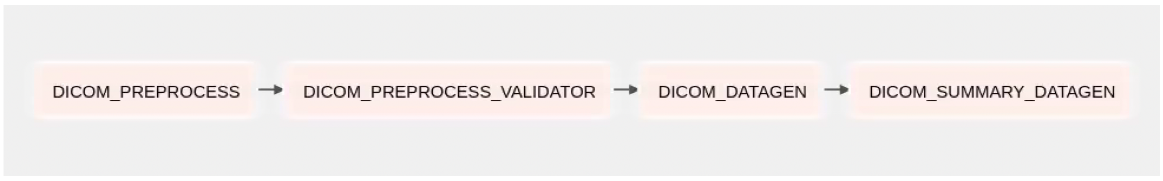
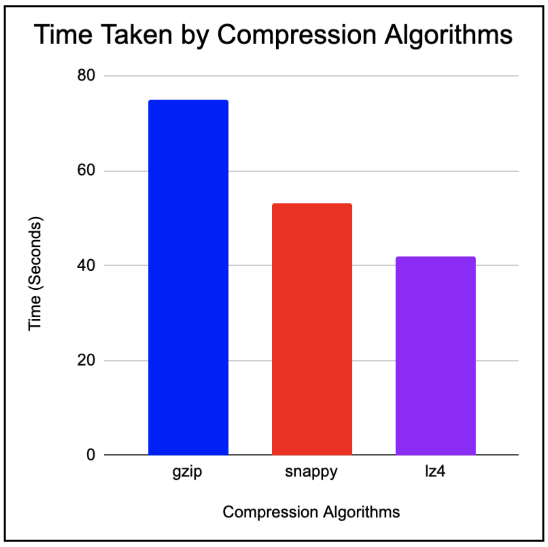


**Fig 12: DAGS as in Apache Airflow**

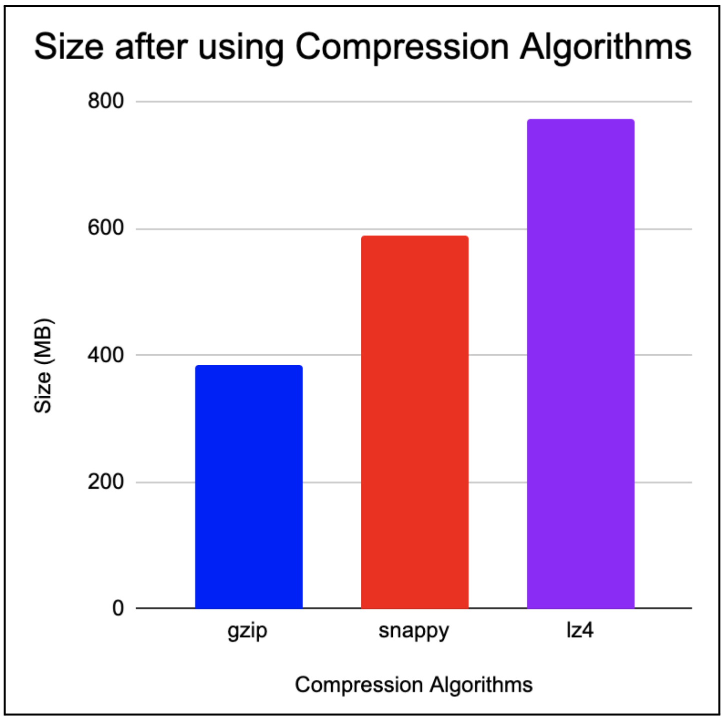
# CHAPTER 4 :EXPERIMENT AND RESULT ANALYSIS

The programs were then executed on a spark cluster, which had a master and 29 worker nodes. The results discussed are a brief idea of the actual run as the details are confidential and are subject not to be disclosed in any manner possible. Here we shall discuss the space efficiency, the time efficiency of the codes and a brief discussion on how the approaches used have contributed to better results.

The first process of data  preprocessing took time of 6 hours and 27 minutes to process a dataset of approximately 75 Tb in size.This is fairly impressive and proves the code is efficient. Also not to forget our processing power is also considerably high using a cluster with 29 worker nodes. The data after removing values was compressed to a size of approximately 25 Tb which is fairly impressive again. This shall reduce the cost of storing a huge dataset by far. The compression chosen for the same was snappy as it is both mediocre time and space efficient as compared to other algorithms offering a balance in between. Here are the results for a small dataset of 1.2 GB:



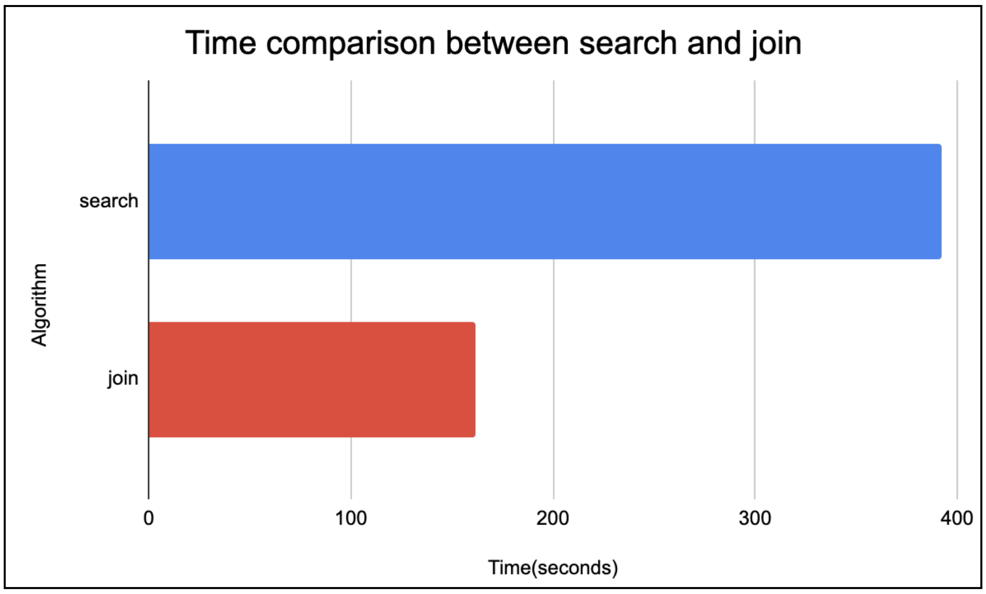
**Graph 1: Time Taken by different compression Algorithms**



**Graph 2: Sizes of files after using compression algorithms.**

Hence Snappy was chosen as it provides a fine balance among all the three compression algorithms.

Next we proceeded to the Validation process. The validation process had to be made such that it takes way less time than datagen itself. Having run the validation process the time taken was checked which came out to be nearly 4 hours and 11 minutes which is fairly fast. This speed was obtained owing to the optimisation done by using joins instead of individual searches. On a dataset of 25 GB the improvement was from 6 minutes 32 seconds down to 2 minutes 42 seconds, which is a massive improvement. Hence the improvement was nearly 60%.[5]



**Graph 3: Time comparison between two algorithms checked**

Following this we come to the datagen process where we add required data to the dataset. This process was again fairly long taking 5 hours 49 minutes owing to various joins done in the data with the existing dataset. Once again the validation of this data was done but this time it took only 1 hour and 17 minutes making it fast and efficient.

# CHAPTER 5 : CONCLUSIONS

## 5.1 Conclusions

In this project, we used Spark to create a data creation pipeline capable of compressing data to 25 TB from 75 TB of raw data. This pipeline was capable of efficiently processing massive amounts of data and transforming it into a parquet format suitable for downstream analysis. Overall, this study emphasises the significance of having an efficient data processing pipeline when dealing with enormous amounts of data. The outcomes of this research illustrate Spark's strength and adaptability for huge data processing workloads. We were able to manage massive amounts of data and provide valuable insights that may guide business choices by exploiting Spark's distributed processing capabilities.Spark along with use of Cluster consisting of one master node and 29 worker nodes allowed us to process this massive data rather quickly. The use of Apache Airflow also enabled us to construct a pipeline that is automated, i.e., runs scripts one after another without any human intervention but only using a few config files containing process details, making the process even more efficient and automated. Parquet format appeared as more efficient over others as it was able to compress the data and could be written simultaneously by multiple worker nodes at one go. Making the process fast and space efficient at the same time. The use of snappy algorithm is however debatable as there exists gzip which is faster and lz4 that is more efficient however snappy is a balance of the two. The choice of the algorithm is highly use case dependent.

## 5.2 Future Scope

The data pipeline seems to be efficient but still has a scope for improvement. This would require more testing on the actual dataset. Also the use of parquet is debatable as data in parquet format cannot be updated periodically. Everytime there is a data update the parquet must be written from scratch. Hence more formats must be explored to ensure speed, space and flexibility at the same time.

# REFERENCES

[1] Inoubli, Wissem & Aridhi, Sabeur & Mezni, Haithem & Maddouri, Mondher & Mephu Nguifo, Engelbert. (2018). A Comparative Study on Streaming Frameworks for Big Data.

[2] N. Renugadevi, S. Saravanan, C.M. Naga Sudha, Revolution of Smart Healthcare Materials in Big Data Analytics, Materials Today: Proceedings, 2021,

[3] Kumar, S., Singh, R. K., & Aggarwal, R. (2017). Big data processing using Apache Spark in Hadoop environment. Journal of Big Data Analytics in Transportation, 1(1), 23-38. doi:10.1007/s42421-017-0003-9

[4] Zaharia, Matei & Xin, Reynold & Wendell, Patrick & Das, Tathagata & Armbrust, Michael & Dave, Ankur & Meng, Xiangrui & Rosen, Josh & Venkataraman, Shivaram & Franklin, Michael & Ghodsi, Ali & Gonzalez, Joseph & Shenker, Scott & Stoica, Ion. (2016). Apache spark: A unified engine for big data processing. Communications of the ACM. 59. 56-65. 10.1145/2934664.

[5] https://stackoverflow.com/questions/46940771/difference-between-a-looku p-and-a-join-in-spark