# Comparative Study between Local and Spark Implementation of K-Means Clustering with Performance Analysis

Project Synopsis Report submitted in partial fulfillment of

The requirements for the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE

Of

WEST BENGAL UNIVERSITY OF TECHNOLOGY

By


| | |
|---|---|
| NEHA KUMARI | Roll no :10900115061 |
| SHRAVANI ROY | Roll no :10900115087 |
| SOUMYADEEP DEWAN | Roll no:10900115098 |


Under the guidance of

Dr.Piyali Chatterjee

DEPARTMENT OF COMPUTER SCIENCE



**NETAJI SUBHASH ENGINEERING COLLEGE**
**TECHNO CITY, GARIA, KOLKATA – 700 152**


2018-19

# Certificate

This is to certify that this project synopsis report titled "**Comparative Study between Local and Spark Implementation of K-Means Clustering with Performance Analysis**" submitted in partial fulfillment of requirements for award of the degree Bachelor of Technology (B. Tech) in Computer Science and Engineering of West Bengal University of Technology is a faithful record of the original work carried out by,

| | | |
|---|---|---|
| **Neha Kumari** , | **Roll no**,10900115061, | **Regd. No.** 151090110061, 2015-19 |
| **Shravani Roy** | **Roll no**.10900115087, | **Regd. No.** 151090110088, 2015-19 |
| **Soumyadeep Dewan** | **Roll no**.10900115098, | **Regd. No.** 151090110099, 2015-19 |

under my guidance and supervision.

It is further certified that it contains no material, which to a substantial extent has been submitted for the award of any degree/diploma in any institute or has been published in any form, except the assistances drawn from other sources, for which due acknowledgement has been made.

---

Dr. Piyali Chatterjee
HOD, Department of CSE
Netaji Subhash Engineering College
Garia, Kolkata - 700152

---

Dr. Piyali Chatterjee
Associate Professor,Dept of CSE
Netaji Subhash Engineering College
Garia, Kolkata - 700152

# Declaration

We hereby declare that this project report titled "**Comparative Study between Local and Spark Implementation of K-Means Clustering with Performance Analysis"** is our own original work carried out as a undergraduate student in Netaji Subhash Engineering College except to the extent that assistances from other sources are duly acknowledged.

All sources used for this project report have been fully and properly cited. It contains no material which to a substantial extent has been submitted for the award of any degree/diploma in any institute or has been published in any form, except where due acknowledgement is made.

| **Student's names:** | **Signatures:** | **Dates:** |
|---|---|---|
| Neha Kumari | …….. | …… |
| Shravani Roy | …….. | …… |
| Soumyadeep Dewan | …….. | …… |

# Certificate of Approval

We hereby approve this dissertation titled **Comparative Study between Local and Spark Implementation of K-Means Clustering with Performance Analysis** carried out by:

| | | |
|---|---|---|
| **Neha Kumari** , | **Roll no**,10900115061, | **Regd. No.** 151090110061, 2015-19 |
| **Shravani Roy** | **Roll no**.10900115087, | **Regd. No.** 151090110088, 2015-19 |
| **Soumyadeep Dewan** | **Roll no**.10900115098, | **Regd. No.** 151090110099, 2015-19 |

under the guidance of **Dr. Piyali Chatterjee** of Netaji Subhash Engineering College, Kolkata in partial fulfillment of requirements for award of the degree Bachelor of Technology (B. Tech) in **Computer Science and Engineering** of Maulana Abul Kalam Azad University of Technology.

**Date:**……..

**Examiners' signatures:**

1. …….

2. …….

3. …….

# Acknowledgement

We would like to thank our mentor, Dr. Piyali Chatterjee for giving us such a highly informative and interesting project topic to work on. Implementing a ML algorithm on Spark is not only a project which would be good as a final year project but we are sure that this project would give us a clinical edge in our professional careers as well. Although working on Spark is beyond interesting and enjoyable, beginners like ourselves face deadlocks where it is very easy to give up. Our mentor assured us that we would be able to complete the project and kept the motivation running in the group. Ma'am provided us the arsenal required to complete the project and we did what we could with that. We would also like to extend our gratitude to Professor Soumyendu Sekhar Bandyopadhyay for familiarising us with Pyspark and Spark in general. Sir would often send us e-books based on our topic and in all honesty, those e-books were like the Bible, something which we could refer to in the darkest of times. We are grateful for his precious time and guidance throughout.

**Neha Kumari**

**Shravani Roy**

**Soumyadeep Dewan**

**Dated:**……

# Abstract

*There has been a tremendous growth in data the past decade. Handling such massive data is challenging and presently is of utmost importance.The amount of data produced by us from beginning of time till 2003 was 10 gigabytes. Clustering can serve as a solution, it divides the data into smaller groups based on the level of similarity among the objects. K-Means is one of the most popular and robust clustering algorithm.Also,the ever increasing data also mandates a big data analytics framework. This paper discusses two of the comparison of -Python and the recently introduced Apache Spark – both of which provide a processing model for analyzing big data. Although both of these options are based on the concept of Big Data, their performance varies significantly.This is what makes these two options worthy of analysis with respect to their variability and variety in the dynamic field of Big Data. In this paper we compare these two implementations along with providing the performance analysis using a standard machine learning algorithm for clustering (K-Means).*

*Spark is known as the Swiss army knife of Big Data Analytics and is revolutionizing Big Data.*

## Table of Contents

# Chapter I: Introduction

Big Data is taking the world by storm. Due to widespread usage of many computing devices such as smartphones, laptops, wearable computing devices; the data processing over the internet has exceeded more than the modern computers can handle.It is stated that almost 90% of today's data has been generated in the past 3 years. Every organization or company, be it in the healthcare, manufacturing, automobile or software industry, needs to manage and analyze the large data volumes that it generates.The ever increasing data is needed to be managed efficiently in terms of storage, processing, retrieval, etc.

Clustering, an unsupervised data mining technique which can be effectively applied in such circumstance. It divides the data into smaller groups called clusters and the elements of one cluster are similar to each other, whereas dissimilar from the objects of other clusters. Clustering is used in exploration of data to find thes shape of data set, and also in detection of anomalies.Moreover, clustering the data, prior to applying the analytics reduces computational cost. K-Means is one of the most popular and widely used clustering algorithm.
The need for big data analytics frameworks can be seen when implementing this algorithm on large datasets. This mandates a framework that uses the existing CPU in the local system in a distributed environment.Among the most prominent tools that achieve this task is Hadoop, an open source platform that provides excellent data management provisions. Its primary use is to facilitate the processing of very large datasets in a distributed computing environment using components like the Hadoop Distributed File System (HDFS), MapReduce, HBase, Hive etc.However, MapReduce programs are very much sensitive to iteration, as in each round the data is written back to the file system. Multiple
read and writes to the file system increases the IO cost. K-Means algorithm is iterative in nature. So, MapReduce based K-Means is quite costly. Moreover, to get the optimal number of clusters we need to execute the algorithm multiple times.As we are dealing with big data, complexity of the process
further increases.
This paper compared and evaluated execution times of existing big data processing framework, Apache Spark and the Scikit-learn ML library for Python. This paper explored a detailed evaluation on the parallel execution of K-Means Clustering in both frameworks. Comparison of performance on different metric has been presented under experimental result.
.
Apache Spark is revolutionizing Big Data- similar to the way the smartphone have changed the way we communicate-far beyond its original aim of cellular telephony. The fulcrum of the mobile revolution has been portability but what made the smartphone a massive consumer product is the ability to have one device perform multiple tasks efficiently with the potential to easily build and use a diverse range of applications.Ultimately, with the smartphone we have a general platform that has changed the way we communicate, socialize, work, and play. The smartphone has not only replaced older technologies but combined them in a close-knit manner that led to new types of user experiences. Applying this analogy to the Big Data space – Apache Spark seeks to be the general platform that is changing the way we work and understand data.

The genesis for Apache Spark was to provide the flexibility and extensibility of Hadoop MapReduce at significantly faster speeds. The problem dates back to the inefficiencies of running machine learning algorithms at UC Berkeley. But this was not about time lost (due to long running executions) or inefficiencies (e.g. errors in long running jobs only revealed themselves at late stages, requiring fixing and re-running the entire job), a crucial component is that long running queries interrupt your train of thought. This is particularly important with data exploration because, after this short period of time, you start losing your train of thought. Faster response times are not just about efficiencies, but making it easier for data explorers to retain and test their connections and hypothesis.
Spark is an integrated platform for data science. Spark was conceived on the need for iterative applications or interactive queries. But it quickly changed course when it was realized that Spark's value extended well beyond high performance machine learning.

With MLlib, Spark users can utilize a scalable machine learning library to build predictive models such as Toyota's Customer 360 Insights Platform. GraphX enables Spark users to interactively build, transform, and reason about graph structured data at scale such as how Alibaba solves their complex network problem. Spark Streaming provides the ability to build powerful streaming applications such as how Netflix solves real-time data monitoring. In addition, with Spark SQL and DataFrames, Spark users can build a just-in-time data warehouse for business analytics.

Spark ecosystem is built on top of the Spark Core, the underlying general execution engine for the Spark platform. It provides in-memory computing capabilities to deliver speed and a generalized execution model to support a wide variety of applications. What makes Spark even more powerful is that you can use your language of choice – Scala, Python, Java, SQL, or R – so your focus is on making sense of your data instead of learning a new tool and language.Instead of having many disparate pieces of technology often put together in a custom or proprietary fashion, Spark provides you a general platform and the necessary tools so that your focus is on your data problems.

The smartphone replaced many navigation applications due to its embedded GPS and app store applications. It replaced the point and click standalone digital camera with its yearly megapixel improvements. Tasks ranging from note taking to calendar scheduling to social media are now mobile first. The smartphone is at the center of a rich ecosystem serving as a powerful platform that can do it all. While Spark is fast and helps us understand our data, more fundamentally it is about how Spark changes the way we do data engineering and data sciences. It allows us to solve a diverse range of data problems from machine learning to streaming in your language of choice. Analogous to smartphone apps, Spark also includes a diverse library of Spark-packages that allows you to share and extend the platform. Just like how the smartphone revolutionized telephony, Spark is revolutionizing Big Data.

## 1.1 An Introduction to Big Data

We are aware that today we have huge data being generated everywhere from various sources. This data is either being stored intentionally in a structured way or getting generated by machines. But data is of no use until we mine it and try to do some kind of analysis on it, in order to come up with actions based on the analysis outcomes. The act of gathering and storing information for eventual analysis is ages old but it had never been based on such a large amount of data, which is there today. There is a specific term for such voluminous data i.e. "**Big Data**".

Big data is a term that describes the huge volume of data which can be structured and unstructured or semi-structured. But it's not the amount of data which is a concern for the organizations since it is just a storage problem which can be easily addressed by the cheap storage available today. It's what Business get out of the data matters.Big data can be analyzed for insights that lead to better decisions and strategic business moves.

Big Data is an important concept, which is applied to data, which does not conform to the normal structure of the traditional database. Due to widespread usage of many computing devices such as smartphones, laptops, wearable computing devices; the data processing over the internet has exceeded more than the modern computers can handle. According to a recent market report published by Transparency Market Research, the total value of big data was estimated at $6.3 billion as of 2012, but by 2018, it's expected to reach the staggering level of $48.3 billion that's almost a 700 percent increase. Forrester Research estimates that organizations effectively utilize less than 5 percent of their available data. This is because the rest is simply too expensive to deal with.

Big Data is derived from multiple sources. It involves not just traditional relational data, but all paradigms of unstructured data sources that are growing at a significant rate. For instance, machine-derived data multiplies quickly and contains rich, diverse content that needs to be discovered. Another example, human-derived data from social media is more textual, but the valuable insights are often overloaded with many possible meanings. Big data is not merely a data, rather it has become a complete subject, which involves various tools, techniques and frameworks.

## 1.2Attributes of Big Data-5Vs

Big data can be described by the following five characteristics:

**Volume**

The quantity of data that is generated is very important in this context. It is the size of the data which determines the value and potential of the data under consideration and whether it can actually be considered as Big Data or not. The name 'Big Data' itself contains a term which is related to size and hence the characteristic.

Example- A typical PC might have had 10 gigabytes of storage in 2000.Today, Facebook ingests 600 terabytes of new data every day. The smart phones, the data they create and consume; sensors embedded into everyday objects will soon result in billions of new, constantly-updated data feeds containing environmental, location, and other information, including video.

**Velocity**

The term 'velocity' in this context refers to the speed of generation of data or how fast the data is generated and processed to meet the demands and the challenges which lie ahead in the path of growth and development.

Example- High-frequency stock trading algorithms reflect market changes within microseconds machine to machine processes exchange data between billions of devices online gaming systems support millions of concurrent users, each producing multiple inputs per second.

**Variety**

The next aspect of Big Data is its variety. This means that the category to which Big Data belongs to is also a very essential fact that needs to be known by the data analysts. The data that is generated is completely heterogeneous in the sense that it could be in various formats like video, text, database, numeric, sensor data and so on and hence understanding the type of Big Data is a key factor to unlocking its value.

Example-Big Data isn't just numbers, dates, and strings. Big Data is also geospatial data, 3D data, audio and video, and unstructured text, including log files and social media.

**Veracity**

Big Data Veracity refers to the biases, noise and abnormality in data. Is the data that is being stored, and mined meaningful to the problem being analyzed.Knowing whether the data that is available is coming from a credible source is of utmost importance before deciphering and implementing Big Data for business needs.

Example-With many forms of big data, quality and accuracy are difficult to control like Twitter posts with hashtags, abbreviations, typos and colloquial speech. The volume is often the reason behind for the lack of quality and accuracy in the data.

**Value**

When we talk about value, we're referring to the worth of the data being extracted. Having endless amounts of data is one thing, but unless it can be turned into value it is useless. While there is a clear link between data and insights, this does not always mean there is value in Big Data. The most important part of embarking on a big data initiative is to understand the costs and benefits of collecting and analyzing the data to ensure that ultimately the data that is reaped can be monetized.


**Fig 1.1 The 5Vs of Big Data.**

## 1.3 Big Data Challenges

The major challenges associated with big data are as follows:-

**Storage Medium Issues**
Mechanical disk drives (HDD) and Solid State Disk (SSD) drives are the major trending storage mediums, with hard disk drives forming the basis of the bulk of big data storage. Reliability risk such as overheating and magnetic faults, and disk access overhead has made HDDs undesirable for big data storage, though price per gigabyte is relatively low. SSDs on the other hand can service I/O processing request at a much faster rate than HDDs, because there is no mechanical part, reducing access time hence increase in I/O rate. SSDs are more resistant to physical shock, hence more reliable. The problem with SSDs is the price per gigabyte. The cost of replacing all mechanical drives with SSDs for a big data storage is unreasonably high.

**The Uncertainty of Data Management**
One disruptive facet of big data management is the use of a wide range of innovative data management tools and frameworks whose designs are dedicated to supporting operational and analytical processing. The NoSQL (not only SQL) frameworks are used that differentiate it from traditional relational database management systems and are also largely designed to fulfill performance demands of big data applications such as managing a large amount of data and quick response times. There are a variety of NoSQL approaches such as hierarchical object representation (such as JSON and XML) and the concept of a key-value storage. The wide range of NoSQL tools, developers and the status of the market are creating uncertainty with the data management.

**Getting Data from Big Data Structures**
It might be obvious that the intent of a big data management involves analyzing and processing a large amount of data. There are many people who have raised expectations considering analyzing huge data sets for a big data platform. They also may not be aware of the complexity behind the transmission, access, and delivery of data and information from a wide range of resources and then loading these data into a big data platform. The intricate aspects of data transmission, access and loading are only part of the challenge. The requirement to navigate transformation and extraction is not limited to conventional relational data sets.

**Security**
Security challenges of big data are quite a vast issue that deserves attention. Quite often, big data adoption projects put security off till later stages. And, frankly speaking, this is not too much of a smart move. Big data technologies do evolve, but their security features are still neglected, since it’s hoped that security will be granted on the application level. And what do we get? Both times (with technology advancement and project implementation) big data security just gets cast aside.

## 1.4 Countering the challenges

We discuss two emerging solutions to the challenges faced by Big Data-Apache Hadoop and Apache Spark.

### 1.4.1 Apache Hadoop

Hadoop is an open source distributed processing framework that manages data processing and storage for big data applications running in clustered systems. It is at the center of a growing ecosystem of big data technologies that are primarily used to support advanced analytics initiatives, including predictive analytics, data mining and machine learning applications. Hadoop can handle various forms of structured and unstructured data, giving users more flexibility for collecting, processing and analyzing data than relational databases and data warehouses provide.

Some remarkable benefits of using hadoop platform are:

**Scalable**

Scalability is the main benefit you get from Hadoop. The platform will store your data and also distribute it in large data sets across your servers. What this means is that the platform will give you the ability to run your applications on thousands of nodes while carrying thousands of terabytes. This is the feature that makes Hadoop the ultimate choice for big data management.

**Cost effective**

Hadoop offers a very cost effective storage solution for your organization's ever growing data sets. You no longer need to down-sample data and classify it based on priority. You also don't need to delete the raw data if you don't have to. Hadoop is a cost effective solution for handling big data sets. The massive storage and data distribution feature makes it possible for you to use all your data without being forced to delete the least essential data. Cost savings are better since the cost of storage with Hadoop is much lower.

**Great flexibility**

With Hadoop, you will be able to derive business insights from a range of data sources like email conversations, social media, and clickstream data. This is possible because the platform enables you to access new data sources and easily use different data types. You will be able to work with structured and unstructured data.The platform can be used for recommendation systems, log processing, market campaign analysis, fraud detection and data warehousing.

**Improved speed**

The data storage method on Hadoop is based on the distributed file system. This clearly maps the exact location of data in the cluster. Second, the tools used in data processing are located on the same server that data is located. This leads to faster data processing.When dealing with large data sets, Hadoop makes it possible for you to process terabytes of data efficiently and within minutes. You can process petabytes of data in just hours instead of days.

**Resistant to failure**

When compared to other platforms, Hadoop has a higher fault tolerance. This is because of how the platform works. In its working, when data is sent to a node, the same data is replicated to other nodes within the cluster. This means that in the event of failure in one or more nodes, there will always be a copy available.

**Continuity**

All the aforementioned benefits of using Hadoop result in one main thing, continuity. The reason why many companies suffer is because they lack the data they need to predict the future. The data was often lost when prioritizing which data to keep and which one to delete. This is no longer the case once you embrace Hadoop. The platform enables you to collect and keep all the data in a well-organized manner. This has enabled many companies to store and analyze the high volume of their data successfully.

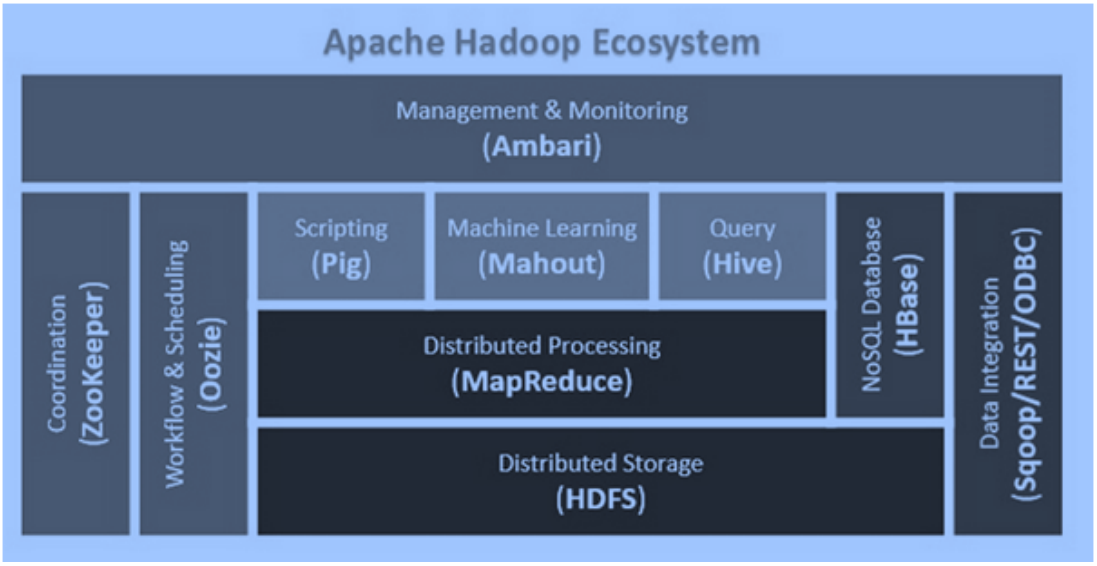


**Fig 1.2 Apache Hadoop Ecosystem.**

### 1.4.2 Apache Spark

Spark, in very simple terms, is a general-purpose data handling and the processing engine that is fit for use in a variety of circumstances. Data scientists make use of Apache Spark to improve their querying, analyses and well as the transformation of data. Tasks most frequently accomplished using Spark include interactive queries across large data sets, analysis, and processing of streaming data from sensors and other sources, as well as machine learning tasks.

From the time it was incepted, it was made sure that most of the tasks happen in-memory. Therefore, it was always going to be faster and much more optimised than other approaches like Hadoop's MapReduce, which writes data to and from hard drives between each stage of processing.

Some benefits of using Spark are as follows:-

**Integration with Hadoop**

Spark's framework is built on top of the Hadoop Distributed File System (HDFS).So it's advantageous for those who are familiar with Hadoop.

**Flexibility and accessibility**

Having such a rich set of APIs, Spark has ensured that all of its capabilities are incredibly accessible. All these APIs are designed for interacting quickly and efficiently with data at scale, thus making Apache Spark extremely flexible. There is thorough documentation for these APIs, and it is written in an extraordinarily lucid and straightforward manner.

**Speed**

Speed is what Spark is designed for. Both in-memory or on disk. A team of Databricks used Spark for the 100TB Benchmark challenge. This challenge involves processing a huge but static data set. The team was able to process 100 TBs of data stored on an SSD in just 23 minutes using Spark. The previous winner did it in 72 minutes using Hadoop. What is even better is that Spark performs well when supporting interactive queries of data stored in memory. In these situations, Apache Spark is claimed to be 100 times faster than MapR.

**Support**

Apache Spark supports most of the famous programming languages including Java, Python, Scala, and R. Spark also includes support for tight integration with a number of storage systems except just HDFS. Furthermore, the community behind Apache Spark is huge, active, and international.

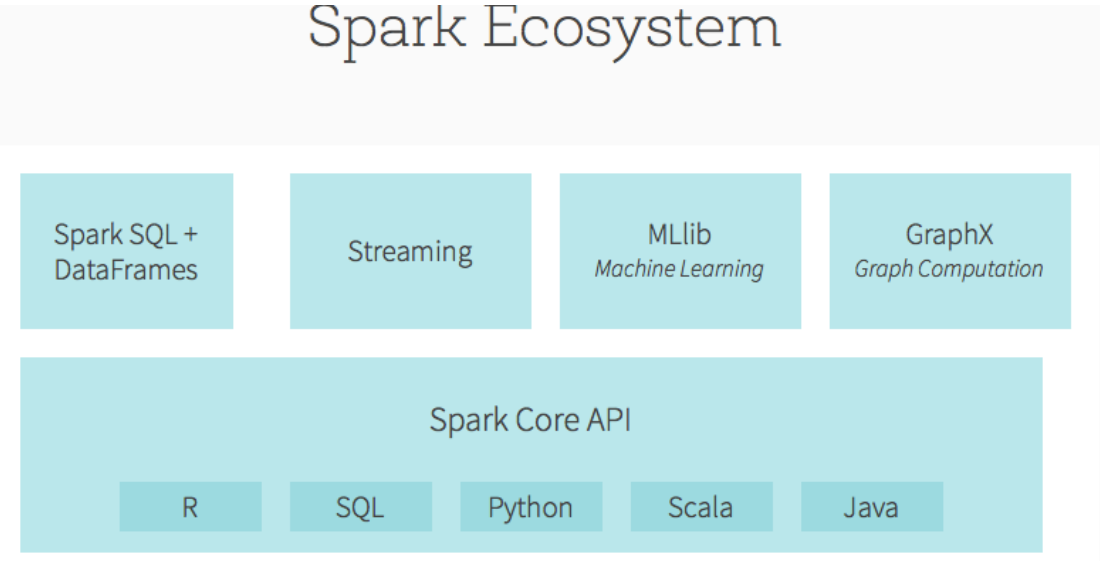


**Fig 1.3 Apache Spark Ecosystem**

## 1.5 Apache Spark over Apache Hadoop

Though they perform many similar tasks there are non-overlappings areas as well which need mentioning. Hadoop as a big data processing technology has proven to be the go to solution for processing large data sets. MapReduce is a great solution for computations, which needs one-pass to complete, but not very efficient for use cases that require multi-pass for computations and algorithms. Each stage in the data processing workflow has one Map and one Reduce phase .To leverage MapReduce solution we need to convert our use case into MapReduce pattern. The Job's output data between each step has to be stored in the file system before the next step can begin. Hence, this approach is slow, due to replication & disk Input/output operations. Also, Hadoop ecosystem doesn't have every component to complete a big data use case. It also requires the integration of several other tools for different big data use cases (like Mahout for Machine Learning and Storm for streaming data processing, Flume for log aggregation).

If you want to do an iterative job, you would have to stitch together a sequence of MapReduce jobs and execute them in sequence. Each of those jobs has high-latency, and each depends upon the completion of previous stages. Spark allows programmers to employ complex, multi-step data pipelines using directed acyclic graph (DAG) pattern. It allows in-memory data sharing across DAGs, so that different jobs can work with the same data without going to disk.

Spark can run on top of Hadoop's distributed file system Hadoop Distributed File System (HDFS) to leverage the distributed replicated storage. Spark can be used along with MapReduce in the same Hadoop cluster or can be used alone as a processing framework. Apache Spark is an alternative to Hadoop MapReduce rather than a replacement of Hadoop. It's not intended to replace Hadoop but it can regarded as an extension to it. In many use cases MapReduce and Spark can be used together where MapReduce job can be used for batch processing and spark can be used for real-time processing.

Spark has several advantages when compared to other big data and MapReduce technologies like Hadoop and Storm. Spark is faster than MapReduce and offers low latency due to reduced disk input and output operation. Spark has the capability of in memory computation and operations, which makes the data processing really fast than other MapReduce.

Unlike Hadoop spark maintains the intermediate results in memory rather than writing every intermediate output to disk. This hugely cuts down the execution time of the operation, resulting in faster execution of task, as more as 100X time a standard MapReduce job. Apache Spark can also hold data onto the disk. When data crosses the threshold of the memory storage it is spilled to the disk. This way spark acts as an extension of MapReduce. Spark doesn't execute the tasks immediately but maintains a chain of operations as metadata of the job called DAG. The action on the DAG happens only when an action operation is called on to the transformation DAG. This process is called as lazy evaluation. This allows optimized execution of the queries on Big Data.

Apache Spark has other features, such as:

- Supports wide variety of operations, compared to Map and Reduce functions.
- Provides concise and consistent APIs in Scala, Java and Python.
- Spark is written in Scala Programming Language and runs in JVM.
- It currently has support in the following programming languages to develop applications in Spark:
    - Scala
    - Java
    - Python
    - R
- Features interactive shell for Scala and Python. This is not available in Java yet.
- It leverages the distributed cluster memory for doing computations for increased speed and data processing.
- Spark enables applications in Hadoop clusters to run up to as much as 100 times faster in memory and 10 times faster even when running in disk.
- It is most suitable for real time decision making with big data.
- It runs on top of existing Hadoop cluster and access Hadoop data store (HDFS), it can also process data stored by HBase structure. It can also run without Hadoop with apache Mesos or alone in standalone mode.
- Apache Spark can be integrated with various data sources like SQL, NoSQL, S3, HDFS, local file system etc.
- Good fit for iterative tasks like Machine Learning (ML) algorithms.
- In addition to Map and Reduce operations, it supports SQL like queries, streaming data, machine learning and data processing in terms of graph.

The truth is that Spark and MapReduce have a symbiotic relationship with each other. Hadoop provides features that Spark does not possess, such as a distributed file system and Spark provides real-time, in-memory processing for those data sets that require it. The perfect big data scenario is exactly as the designers intended—for Hadoop and Spark to work together on the same team.

## 1.6 Apache Spark:Components and Architecture

SparkContext is an independent process through which spark application runs over a cluster. It gives the handle to the distributed mechanism/cluster so that you may use the resources of the distributed machines in your job. Your application program which will use SparkContext object would be known as driver program. Specifically, to run on a cluster, the SparkContext connects to several types of cluster managers (like Spark's own standalone cluster manager, apache Mesos or Hadoop's YARN), which allocate resources across applications. Once connected, Spark takes over executors on distributed nodes in the cluster, which are processes in the distributed nodes that run computations and store data for your application. Next, it sends your application code to the executors through SparkContext. Finally tasks are sent to the executors to run and complete it.

### 1.6.1 Cluster overview

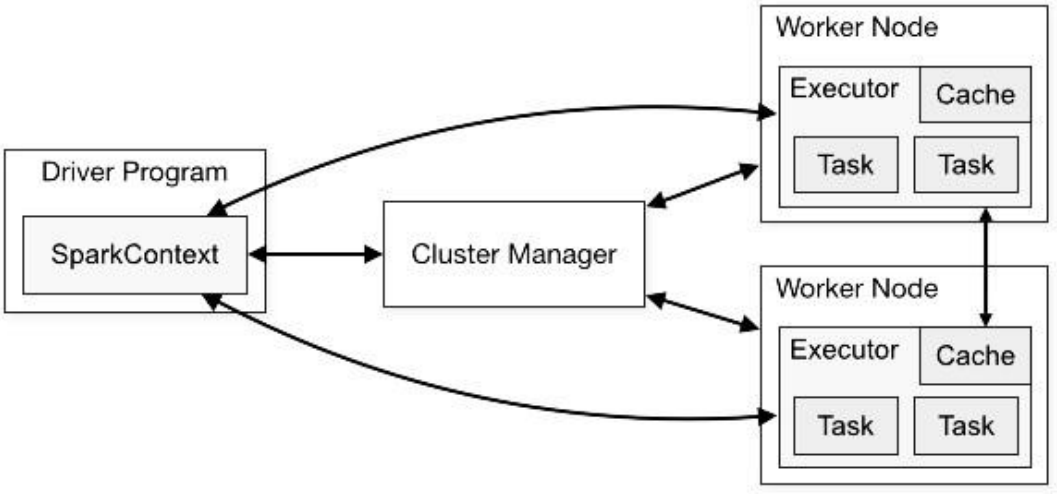


**Fig 1.4 Execution flow in Spark.**

Following are most important takeaways of the architecture:

- Each application gets its own executor processes, which remains in memory up to the duration of the complete application and run tasks in multiple threads. This means each application is independent from the other, on both the scheduling side since each driver schedules its own tasks and executor side as tasks from different applications run in different JVMs.
- Spark is independent of cluster managers that implies, it can be coupled with any cluster manager and then leverage that cluster.
- Because the driver schedules tasks on the cluster, it should be run as close to the worker nodes as possible.

### 1.6.2 Spark Ecosystem

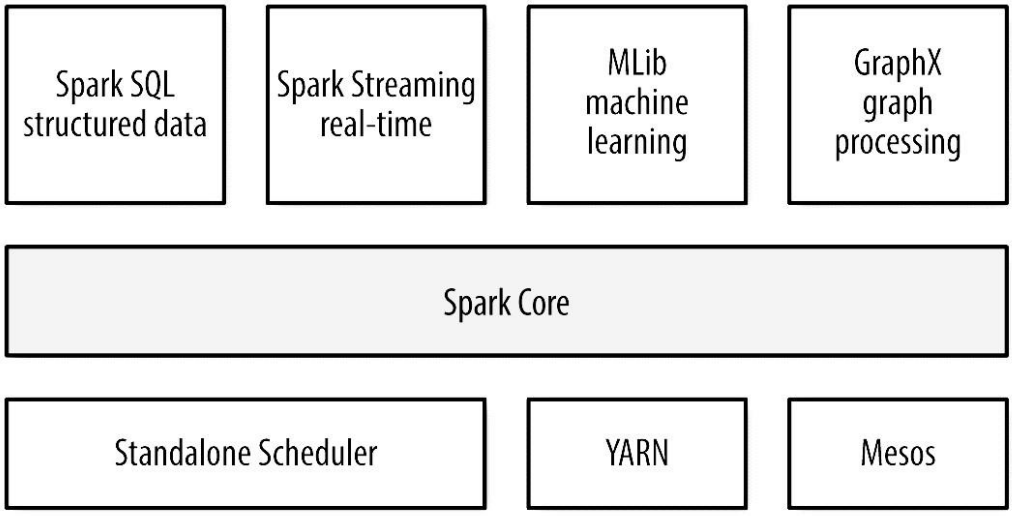


**Fig 1.5 Spark ecosystem**

**Spark Core**

Spark Core is the base of an overall spark project. It is responsible for distributed task dispatching, parallelism, scheduling, and basic I/O functionalities. All the basic functionality of spark core are exposed through an API (for Java, Python, Scala, and R) centered on the RDD abstraction. A 'driver' program starts parallel operations such as map, filter or reduce on any RDD by passing a function to Spark Core, which further schedules the function's execution in parallel on the cluster.

Other than Spark Core API, there are additional useful and powerful libraries that are part of the Spark ecosystem and adds powerful capabilities in Big Data analytics and Machine Learning areas. These libraries include:

**Spark Streaming**

Spark Streaming is a useful addition to the core Spark API. It enables high-throughput, fault-tolerant stream processing of live data streams. It is used for processing real-time streaming data. This is based on micro batch style of computing and processing. The fundamental stream unit is DStream. DStream is basically a series of RDDs, to process the real-time data.


**Fig 1.6 Spark Streaming**

**Spark SQL, DataFrames and Datasets:**

- *SQL*
  Spark SQL exposes spark APIs to run SQL query like computation on large data. A spark user can perform ad-hoc query and perform near real time ETL on a different types of data like (like JSON, Parquet, Database).

- *DataFrames*
  A DataFrame can be considered as a distributed set of data which has been organized into many named columns. It can be compared with a relational table, CSV file or a data frame in R or Python. The DataFrame functionality is made available as API in Scala, Java, Python, and R.

- *Dataset*
  A Dataset is a new addition in the list of spark libraries. It is an experimental interface added in Spark 1.6 that tries to provide the benefits of RDDs with the benefits of Spark SQL's optimized execution engine.

- *Spark MLlib And ML*
  MLlib is collective bunch few handy and useful machine learning algorithms and data cleaning and processing approaches which includes classification, clustering, regression, feature extraction, dimensionality reduction, etc. as well as underlying optimization primitives like SGD and BFGS.

- *Spark GraphX*
  GraphX is the Spark API for graphs and graph-parallel computation. GraphX enhances the Spark RDD by introducing the Resilient Distributed Property Graph.
  A RDD property graph is a directed multi-graph with properties attached with each of its vertex and edge. GraphX has a set of basic operators (like subgraph, joinVertices, aggregateMessages, etc.).Along with operators it has an optimized variant of the Pregel API. GraphX is still under development and many developers are working towards simplification of graph related tasks.

## 1.7 Resilient Distributed Datasets

Resilient Distributed Datasets(RDDs) are a distributed collection of immutable JVM objects which form the backbone of Spark and allow fast calculations.
Decomposing the name RDD:

- Resilient, i.e. fault-tolerant with the help of RDD lineage graph(DAG) and so able to recompute missing or damaged partitions due to node failures.
- Distributed, since Data resides on multiple nodes.
- Dataset represents records of the data you work with. The user can load the data set externally which can be either JSON file, CSV file, text file or database via JDBC with no specific data structure.

Hence, each and every dataset in RDD is logically partitioned across many servers so that they can be computed on different nodes of the cluster. RDDs are fault tolerant i.e. It posses self-recovery in the case of failure.

### 1.7.1 Features of Spark RDD

i. In-memory Computation
Spark RDDs have a provision of in-memory computation. It stores intermediate results in distributed memory(RAM) instead of stable storage(disk).

ii. Lazy Evaluations
All transformations in Apache Spark are lazy, in that they do not compute their results right away. Instead, they just remember the transformations applied to some base data set.
Spark computes transformations when an action requires a result for the driver program. Follow this guide for the deep study of Spark Lazy Evaluation.

iii. Fault Tolerance
Spark RDDs are fault tolerant as they track data lineage information to rebuild lost data automatically on failure. They rebuild lost data on failure using lineage, each RDD remembers how it was created from other datasets (by transformations like a map, join or group By) to recreate itself. Follow this guide for the deep study of RDD Fault Tolerance.

iv. Immutability
Data is safe to share across processes. It can also be created or retrieved anytime which makes caching, sharing & replication easy. Thus, it is a way to reach consistency in computations.

v. Partitioning
Partitioning is the fundamental unit of parallelism in Spark RDD. Each partition is one logical division of data which is mutable. One can create a partition through some transformations on existing partitions.
vi. Persistence

Users can state which RDDs they will reuse and choose a storage strategy for them (e.g., in-memory storage or on Disk).

vii. Coarse-grained Operations
It applies to all elements in datasets through maps or filter or group by operation.
viii. Location-Stickiness

RDDs are capable of defining placement preference to compute partitions. Placement preference refers to information about the location of RDD. The DAGScheduler places the partitions in such a way that task is close to data as much as possible. Thus, speed up computation.

**1.7.2 Spark RDD Operations**

RDD in Apache Spark supports two types of operations:

- Transformation
- Actions

I. Transformation-Spark RDD Transformations are *functions* that take an RDD as the input and produce one or many RDDs as the output. They do not change the input RDD (since RDDs are immutable and hence one cannot change it), but always produce one or more new RDDs by applying the computations they represent e.g. Map(), filter(), reduceByKey() etc.
Transformations are lazy operations on an RDD in Apache Spark. It creates one or many new RDDs, which executes when an Action occurs. Hence, Transformation creates a new dataset from an existing one.

II. Action- An Action in Spark returns final result of RDD computations. It triggers execution using lineage graph to load the data into original RDD, carry out all intermediate transformations and return final results to Driver program or write it out to file system. Lineage graph is dependency graph of all parallel RDDs of RDD.
Actions are RDD operations that produce non-RDD values. They materialize a value in a Spark program. An Action is one of the ways to send result from executors to the driver. First(), take(), reduce(), collect(), the count() is some of the Actions in spark.
Using transformations, one can create RDD from the existing one. But when we want to work with the actual dataset, at that point we use Action. When the Action occurs it does not create the new RDD, unlike transformation. Thus, actions are RDD operations that give no RDD values. Action stores its value either to drivers or to the external storage system. It brings laziness of RDD into motion.

# Chapter 2: LITERATURE REVIEW

This chapter is a review of the literature that is available in the areas under consideration and recent advances in the same. The literature in this area includes work that has been done in the fields of big data, Spark and machine learning.We study this literature with the intention to analyse past work in these areas and understand their limitations in order to engineer a system capable of handling these limitations. The literature survey pertaining to this paper has been presented in four parts:

(i) Work related to K-means clustering, (ii) Work related to Apache Spark, (iii) Case study of Apache Spark.

**i.Work related to K-means clustering.**

In the recent years, many clustering algorithms for big data have been proposed which are based on distributed and parallel computation. Cui, Xiaoli and others proposed optimized big data K-Means using MapReduce in which they claimed to counter the iteration dependence of MapReduce jobs[10]. They used a sequence of three MapReduce (MR) jobs for the purpose. However, in their approach sampling technique is used in the first MR job and in the final MR job the data set is mapped to centroids using the Voronoi diagram. Variety is an important feature in big data according to Laney, D.[11],thus using sampling techniques is questionable when applied to huge data sets in maintaining the quality of clustering.

Another parallel K-Means algorithm has been presented by Zhao based on MR, but the initial seed selection is random[12]. Moreover, the algorithm is iterative, and mapreduce jobs are influenced by multiple iterations.Bahmani et have also presented MapReduce based K-Means clustering algorithm[13]. Cai et. al modified the K-Means clustering algorithm to deal with large scale heterogeneous data sets[14].

The authors of [1] presents a novel K-means clustering using Spark which automates the number of clusters to deal with big data whereas the authors of [2] discusses the implementation of the K-Means Clustering Algorithm over a distributed environment using Apache Hadoop.

**ii.Work related to Big Data Analytics and Apache Hadoop**

The authors of [8] have used the MAP function for finding the new cluster centres after which the algorithm implemented in hadoop platform. Yurong Zhong and Dan Liu provides a time complexity analysis for the same and concludes it as an effective process[8]. Zaharia et al. pointed out that mapreduce has been very successful on implementing machine learning algorithms on large datasets[15].Even so, Apache Hadoop has its own limitations. A paper by [9] shows a comparative study between Apache Spark and Hadoop while running K- means algorithm using both the frameworks. The time difference is massive, thus [9] declares Spark as our clear winner as it facilitates batch processing as well as streaming.

**iii.Work related to Big Data Analytics & Apache Spark**

The architecture and utility of Apache Spark was first presented to the community by the authors of [3]. It gives a brief overview regarding the programming model, which includes RDDs, parallel computing etc. It also introduces a few implementations in the environment. [4] introduces Apache Sparks machine learning library, MLlib. This includes its core features as well as the different components constituting the MLlib library.The authors of [5] present a comprehensive results based on their evaluation of by

analyzing twitter data using Sparks framework and predicting the outcome of 2016 Presidential Election. The massive size of the twitter dataset used highlights the significance of using a tool like Spark.

**iv.Case Study-Apache Spark in E-commerce industry**

As many e-commerce companies started collecting huge amount of data, they are using Spark for getting value out it. Companies wanted to study on the kind of products its customer is buying so that recommendation and new products can be shown to customers. Using Spark, the purchase transaction details can be passed to streaming clustering algorithms. The results obtained by these algorithms can be combined with data from social media, reviews, feedback, and customer comment to give better recommendation to customer. Shopify is stores partnership company which wanted to understand what its customers are selling so that it can plan to do partnership with such companies of which the customer selling more product. However without using Spark they were unable to achieve it. Upon using the Spark capabilities they were able to process 67 Million records in a minutes and it has created a list of companies it can probably do partnership. [6] EBay one of the giant e-commerce company has used Apache Spark to provide better and matching offers to targeted customers and also to improve the customer experience. With the help of Hadoop YARN, generic tasks have been run by EBay which used clusters in the range from 2000 to 20,000 nodes with help of 1000 TB RAM. [7]

# CHAPTER 3: PROPOSED WORK

## 3.1 Algorithm of K-Means

We are given a data set of items, with certain features, and values for these features (like a vector). The task is to categorize the items into k groups of similarity. To calculate that similarity, we will use the euclidean distance as measurement.

**Flowchart**

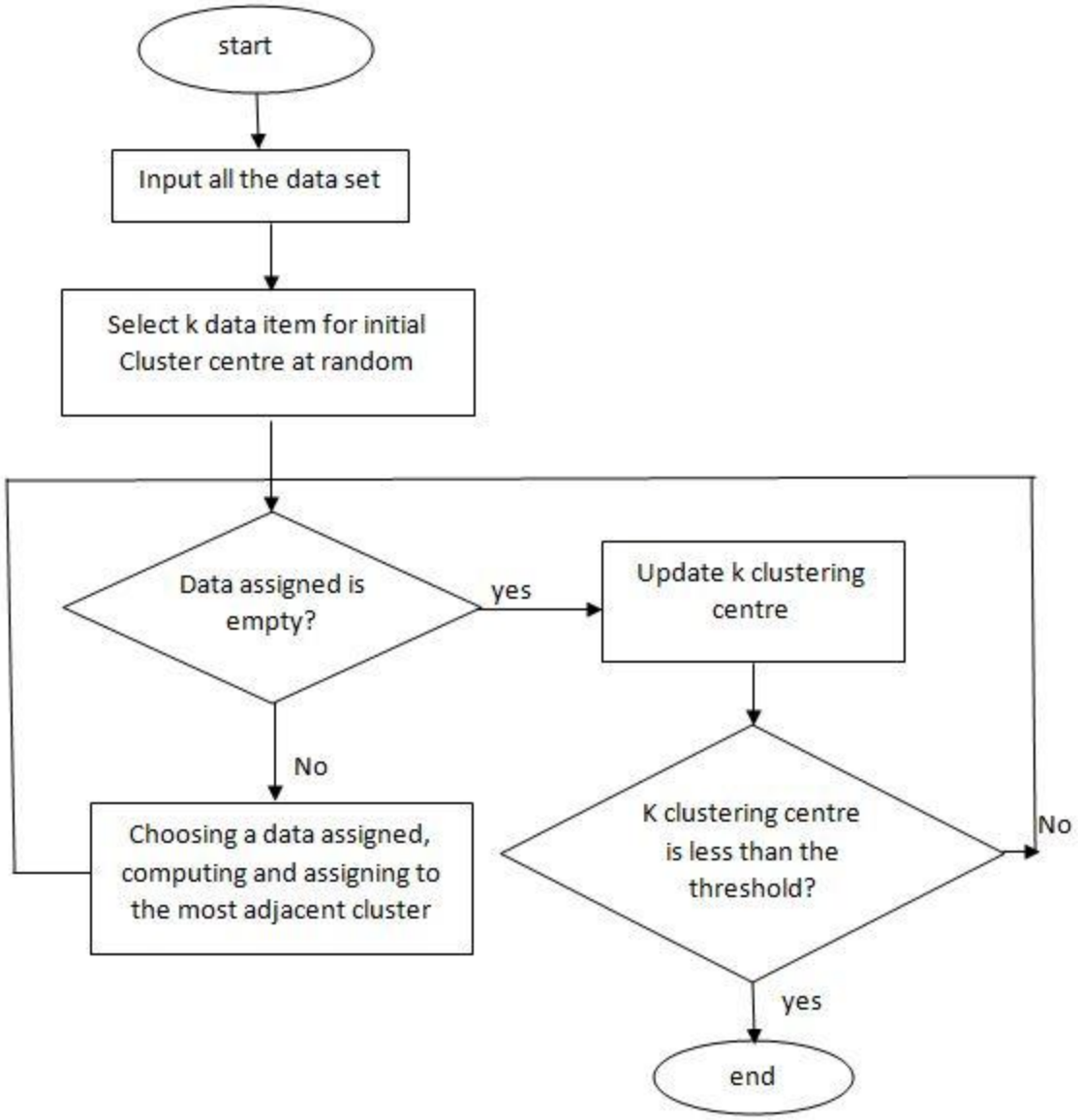


**Fig 1.7 Workflow of K-Means Clustering Algorithm**

## Algorithm

**Initialization step:**Initialize k centroids

**Do**

**Assignment step:** Assign each data point to its closest centroid

**Re-estimation step:** Re-compute centroid(cluster centers)

**While(**there are still changes in the centroid)

## Internal Working of K-means

1 Initial centroids are often chosen randomly.

-Clusters produced vary from one run to another

2. The centroid is (typically) the mean of the points in the cluster.

3.‘Closeness’ is measured by Euclidean distance, cosine similarity, correlation, etc.

4. K-means will converge for common similarity measures.

5. Most of the convergence happens in the first few iterations.

# CHAPTER 4: EXPERIMENTS AND RESULTS

## 4.1 Data Description

We have used the Titanic dataset and prior to discussing the implementation it is very important to have an idea about the dataset we have worked with. The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew.
One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.
The training dataset contains several records of the passengers on the Titanic. It has 11 capturing information about the passengers given below in table 1.

| Attribute | Description |
|---|---|
| Pclass | Passenger Class<br>1 = 1st Class<br>2 = 2nd Class<br>3 = 3rd Class |
| Name | Name of the Passenger |
| Sex | Passenger's Gender<br>Male / Female |
| Age | Passenger's Age |
| SibSp | Number of Siblings/Spouses Aboard |
| Parch | Number of Parents/Children Aboard |
| Ticket | Ticket Number of Passenger |
| Fare | Passenger Fare(in British Pounds) |
| Cabin | Cabin Assigned |
| Embarked | Port of Embarkation<br>C = Chertown<br>Q = Queenstown<br>S = Southampton |
| Survived | Whether passenger survived or not(0 = No & 1 = Yes) |

**Table 1. Data description of the Titanic dataset**

## 4.2 Local Implementation using Python

We start by dropping the 'Survived' column from the dataset and making it unlabelled.As given in Fig 1.8, we used the following Python packages: pandas, NumPy, scikit-learn, Seaborn and Matplotlib.

```
import pandas as pd
import numpy as np
from sklearn.cluster import KMeans
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

**Fig 1.8**

The dataset is then loaded as shown in figure 1.9

```
train_url = "\\Users\\Hp\\Downloads\\train.csv"
train = pd.read_csv(train_url)
test_url = "\\Users\\Hp\\Downloads\\test.csv"
test = pd.read_csv(test_url)
```

**Fig 1.9**

Some samples from the train and test data frames are  are given below in fig 2.0.

```
***** Train_Set *****
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S


***** Test_Set *****
   PassengerId  Pclass                                          Name     Sex  \
0          892       3                              Kelly, Mr. James    male
1          893       3              Wilkes, Mrs. James (Ellen Needs)  female
2          894       2                     Myles, Mr. Thomas Francis    male
3          895       3                              Wirz, Mr. Albert    male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked
0  34.5      0      0   330911   7.8292   NaN        Q
1  47.0      1      0   363272   7.0000   NaN        S
2  62.0      0      0   240276   9.6875   NaN        Q
3  27.0      0      0   315154   8.6625   NaN        S
4  22.0      1      1  3101298  12.2875   NaN        S
```

**Fig 2.0**

Pandas' describe() method is used to fetch some initial statistics about the dataframes. Fig 2.1 shows these initial statistics

```
***** Train_Set *****
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200


***** Test_Set *****
       PassengerId      Pclass         Age       SibSp       Parch        Fare
count   418.000000  418.000000  332.000000  418.000000  418.000000  417.000000
mean   1100.500000    2.265550   30.272590    0.447368    0.392344   35.627188
std     120.810458    0.841838   14.181209    0.896760    0.981429   55.907576
min     892.000000    1.000000    0.170000    0.000000    0.000000    0.000000
25%     996.250000    1.000000   21.000000    0.000000    0.000000    7.895800
50%    1100.500000    3.000000   27.000000    0.000000    0.000000   14.454200
75%    1204.750000    3.000000   39.000000    1.000000    0.000000   31.500000
max    1309.000000    3.000000   76.000000    8.000000    9.000000  512.329200
```

**Fig 2.1**

It is important to note that K-Means does not support missing values in the data and Thus, the missing values present need to be handled. The missing values in both the datasets are calculated and results are displayed in the Fig 2.2.

```
*****In the train set*****
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64


*****In the test set*****
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

**Fig 2.2**

In the training set, in the columns Age, Cabin and Embarked, there are missing values and in the test set, the Age and Cabin columns contain missing values.As shown in Fig 2.3, imputation is performed by replacing the missing values with their mean using the fillna() function provided by Pandas.

```
train.fillna(train.mean(), inplace=True)
test.fillna(test.mean(), inplace=True)
```

**Fig 2.3**

We again check for missing values and the results are displayed in Fig 2.4 and 2.5

- **Training Data**

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

**Fig 2.4**

- **Test Data**

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age              0
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          327
Embarked         0
dtype: int64
```

**Fig 2.5**

There are still some missing values in the Cabin and Embarked columns because these values are non-numeric.

We analyze the various features and start by finding out which of them are categorical and which of them are numeric.

- **Categorical:** Survived, Sex, and Embarked.
- **Ordinal:** Pclass.
- **Continuous:** Age, Fare.
- **Discrete:** SibSp, Parch.

The two features, Ticket and Cabin are not listed above because Ticket is a mix of numeric and alphanumeric data types and Cabin is alphanumeric.Next, we find out the survival count of passengers related to PClass, Sex and SibSp.

Survival Count with respect to PClass is shown Fig 2.6:

| | Pclass | Survived |
|---|---|---|
| **0** | 1 | 0.629630 |
| **1** | 2 | 0.472826 |
| **2** | 3 | 0.242363 |

**Fig 2.6**

Similarly we find the Survival Count with respect to Sex and SibSp and the results are shown in Fig 2.7 and 2.8.

| | Sex | Survived |
|---|---|---|
| **0** | female | 0.742038 |
| **1** | male | 0.188908 |

**Fig 2.7**

| | SibSp | Survived |
|---|---|---|
| **1** | 1 | 0.535885 |
| **2** | 2 | 0.464286 |
| **0** | 0 | 0.345395 |
| **3** | 3 | 0.250000 |
| **4** | 4 | 0.166667 |
| **5** | 5 | 0.000000 |
| **6** | 8 | 0.000000 |

**Fig 2.8**

We then plot a graph of "Age VS Survived" as shown in the Fig 2.9

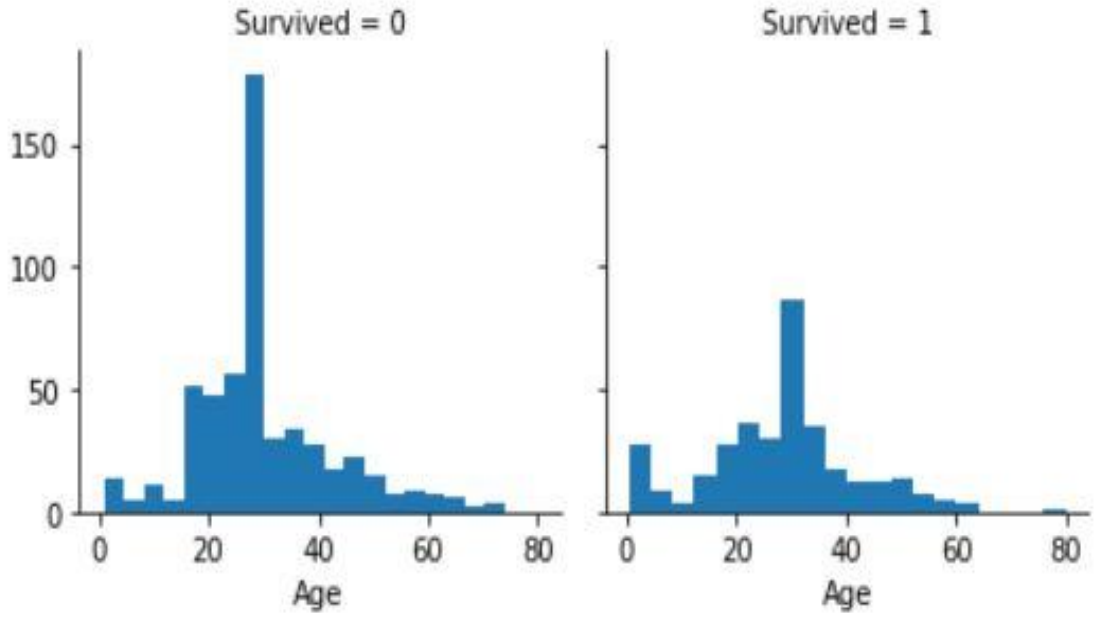


**Fig 2.9**

We show how PClass and Survived are related to each other using a graph in Fig. 3.0.

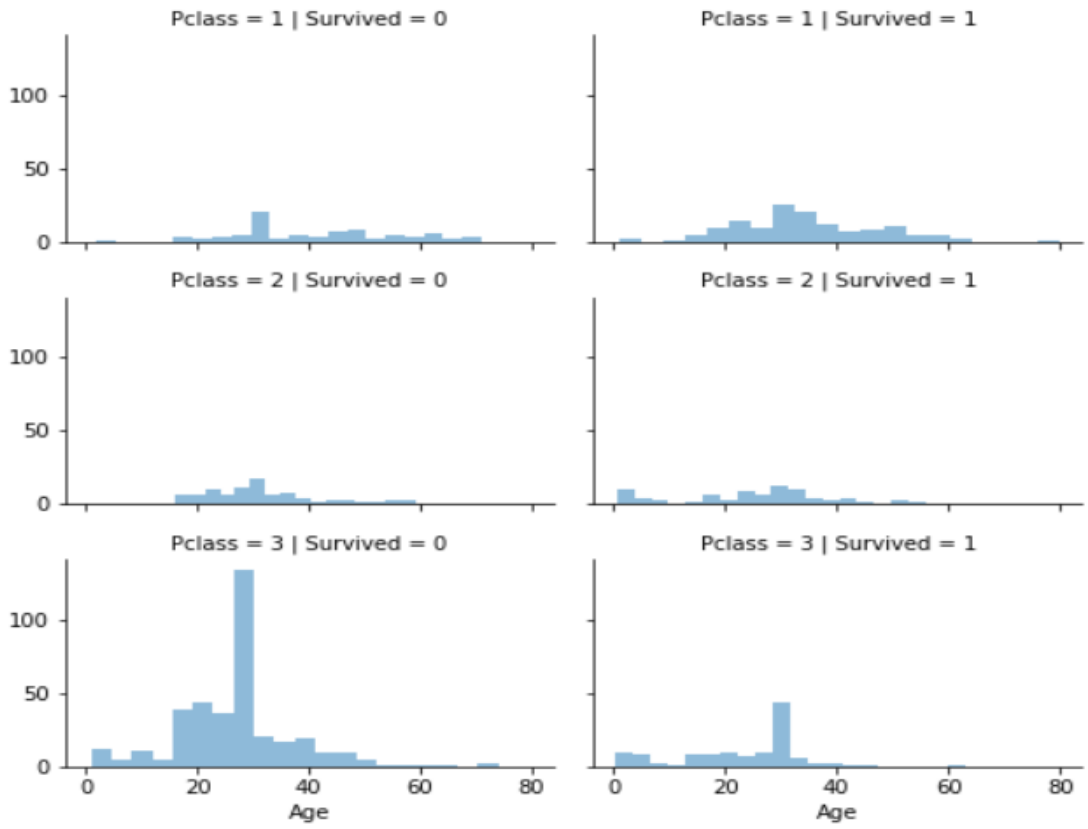


**Fig. 3.0**

It is evident that not all the feature values are of the same type. It is necessary for us to feed only numeric data into the K-Means model for producing better results.

The list of non numeric features are:

- Name
- Sex
- Ticket
- Cabin
- Embarked

Features like Name, Ticket, Cabin and Embarked do not have any impact on Survival Status so these attributes are dropped.

We, then convert the Sex feature into a numeric one

```
labelEncoder = LabelEncoder()
labelEncoder.fit(train['Sex'])
labelEncoder.fit(test['Sex'])
train['Sex'] = labelEncoder.transform(train['Sex'])
test['Sex'] = labelEncoder.transform(test['Sex'])
```

**Fig 3.1**

Finally we start building our K-Means model after dropping the Survived column. Fig 3,2 contains the code used for removing the Survived column and the building process of K-means algorithm, along with the values of its various parameters.

```
X = np.array(train.drop(['Survived'],1).astype(float))
y = np.array(train['Survived'])
kmeans = KMeans(n_clusters=2) # passenger records are clustered into 2: Survived or Not survived
kmeans.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=2, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

**Fig 3.2**

We determine the performance of the model by looking at the percentage of passenger records that were clustered correctly.

```
correct = 0.
for i in range(len(X)):
    predict_me = np.array(X[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == y[i]:
        correct += 1

print(correct/len(X))
```

```
0.5084175084175084
```

**Fig 3.3**

Upon executing the code given in Fig 3,3, our model was able to cluster correctly with 50% accuracy.

We tweak certain parameters from scikit-learn as shown in Fig 3.4.

```
kmeans = kmeans = KMeans(n_clusters=2, max_iter=600, algorithm = 'auto')
```

**Fig 3.4**

Next, we scale the value of the different features that we are feeding the mode as shown in Fig 3.5.

```
scaler = MinMaxScaler()
X_scaled = scaler.fit_transform(X)
```

**Fig 3.5**

Upon doing this, we again test the performance of this model using the code given below in Fig 3.6

```
correct = 0.
for i in range(len(X)):
    predict_me = np.array(X[i].astype(float))
    predict_me = predict_me.reshape(-1, len(predict_me))
    prediction = kmeans.predict(predict_me)
    if prediction[0] == y[i]:
        correct += 1

print(correct/len(X))
```

```
0.6262626262626263
```

**Fig 3.6**

According to Fig 3.6, the accuracy of our model increases by 12%.

## 4.3 Implementation using Apache Spark

The entry point into all the functionality in Spark is the SparkSession class. The first step is to create an RDD and then load the data into Spark dataframe using the load function provided in SparkSession as shown in Fig 3.7

```
from pyspark.sql import SparkSession

spark=SparkSession\
.builder\
.appName('Examine data about passengers on the Titanic')\
.getOrCreate()

rawData=spark.read\
.format('csv')\
.option('header','true')\
.load('\\Users\\Hp\\Downloads\\train.csv')
```

**Fig 3.7**

Fig 3.8 displays some samples provided to us in tabular format.

| | PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 3 | Braund, Mr. Owen Harris | male | 22 | 1 | 0 | A/5 21171 | 7.25 | None | S |
| **1** | 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Th... | female | 38 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| **2** | 3 | 1 | 3 | Heikkinen, Miss. Laina | female | 26 | 0 | 0 | STON/O2. 3101282 | 7.925 | None | S |
| **3** | 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35 | 1 | 0 | 113803 | 53.1 | C123 | S |
| **4** | 5 | 0 | 3 | Allen, Mr. William Henry | male | 35 | 0 | 0 | 373450 | 8.05 | None | S |

**Fig 3.8**

We use the Spark Sql feature to define a table with the features necessary to us. These are the features we need to work with are:

- Survived
- PClass
- Sex
- Age
- Fare
- Embarked

The features Name, SibSp, Parch, Ticket and Cabin are dropped as sas they do not play an important role on the Survival status. Fig 3.9 shows the way in which these columns can be dropped.

```
from pyspark.sql.functions import col

dataset=rawData.select(col('Survived').cast('float'),
                       col('PClass').cast('float'),
                       col('Sex'),
                       col('Age').cast('float'),
                       col('Fare').cast('float'),
                       col('Embarked')
                      )
dataset.toPandas().head()
```

**Fig 3.9**

As K-Means does not support missing values, we need to handle them. As shown in Fig 4.0, we use the dropna() function provided by Pandas to remove all the missing values present in our data.

```
dataset=dataset.replace('?',None).dropna(how='any')
```

**Fig 4.0**

As it is very evident not all the features we have are numeric in nature. The Sex and Embarked feature contain categorical data. Since we can only feed numeric data to our K-Means model, it is very important for us to convert these features into numeric ones. As shown in Fig 4.1, we use the StringIndexer to create two new columns Gender and Boarded from the two input columns Sex and Embarked respectively. The Gender column contains values 0.0 for male and 1.0 for female. Similarly the Boarded column contains 0.0 for passengers from Southampton, 1.0 for passengers from Chertown and 2.0 for passengers from Queenstown(Fig 4.2).

```
from pyspark.ml.feature import StringIndexer
#convert categorical data from string to columns with numeric value
dataset=StringIndexer(
    inputCol='Sex',
    outputCol='Gender',
     handleInvalid='keep').fit(dataset).transform(dataset)

dataset=StringIndexer(
    inputCol='Embarked',
    outputCol='Boarded',
    handleInvalid='keep').fit(dataset).transform(dataset)
```

**Fig 4.1**

| | Survived | PClass | Sex | Age | Fare | Embarked | Gender | Boarded |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 3.0 | male | 22.0 | 7.250000 | S | 0.0 | 0.0 |
| **1** | 1.0 | 1.0 | female | 38.0 | 71.283302 | C | 1.0 | 1.0 |
| **2** | 1.0 | 3.0 | female | 26.0 | 7.925000 | S | 1.0 | 0.0 |
| **3** | 1.0 | 1.0 | female | 35.0 | 53.099998 | S | 1.0 | 0.0 |
| **4** | 0.0 | 3.0 | male | 35.0 | 8.050000 | S | 0.0 | 0.0 |

**Fig 4.2**

Now we can drop the columns Sex and Embarked. Upon doing this, we create and array containing all the essential features.VectorAssembler assembles the required features into a single column named features as shown in Fig 4.3 and 4.4. VectorAssembler is a transformer in Spark, it takes in our dataframe and creates a new dataframe with a new column added.

```
from pyspark.ml.feature import VectorAssembler
assembler=VectorAssembler(inputCols=requiredFeatures,outputCol='features')
```

**Fig 4.3**

| | Survived | PClass | Age | Fare | Gender | Boarded | features |
|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 3.0 | 22.0 | 7.250000 | 0.0 | 0.0 | [0.0, 3.0, 22.0, 7.25, 0.0, 0.0] |
| **1** | 1.0 | 1.0 | 38.0 | 71.283302 | 1.0 | 1.0 | [1.0, 1.0, 38.0, 71.2833023071289, 1.0, 1.0] |
| **2** | 1.0 | 3.0 | 26.0 | 7.925000 | 1.0 | 0.0 | [1.0, 3.0, 26.0, 7.925000190734863, 1.0, 0.0] |
| **3** | 1.0 | 1.0 | 35.0 | 53.099998 | 1.0 | 0.0 | [1.0, 1.0, 35.0, 53.099998474121094, 1.0, 0.0] |
| **4** | 0.0 | 3.0 | 35.0 | 8.050000 | 0.0 | 0.0 | [0.0, 3.0, 35.0, 8.050000190734863, 0.0, 0.0] |

**Fig 4.4**

Finally it is time to start building our K-Means model.As shown in Fig 4.5,we use the machine learning module of Spark called MLlib designed to invoke machine learning algorithms on numerical data sets represented in RDD.
MLlib implementation of k-means corresponds to the algorithm called K-Means\\5 which is a parallel version of the original one. The method header is defined as follows:

*KMeans.train(k, maxIterations, initializationMode, runs)*

- K: number of desired clusters
- maxIterations: the maximum number of iterations that the algorithm will perform. The more iterations the more precision in results but the execution time will increase.
- initializationMode: specifies the type of initialization of the algorithm.
- runs: number of times to execute the algorithm

```
from pyspark.ml.clustering import KMeans
#seed determines the cluster centers
kmeans=KMeans(k=8,seed=1)
model=kmeans.fit(transformed_data)
```

**Fig 4.5**

We use model.transform on our training data to get clustered result in a dataframe as shown in Fig 4.6.The Clustering Evaluator helps us to evaluate how well we have clustered the underlying data.

```
from pyspark.ml.evaluation import ClusteringEvaluator

evaluator=ClusteringEvaluator()
silhouette=evaluator.evaluate(clusterData)
```

**Fig 4.6**

Next, we have to evaluate the Silhouette distance which denotes how similar an object is to its cluster. A Silhouette distance of 1 is considered to be ideal. Fig 4.7 shows the value of Silhouette distance calculated.

```
print('Silhouette with squared euclidean distance=',silhouette)
```

```
Silhouette with squared euclidean distance= 0.6074848997883804
```

**Fig 4.7**

After this, we print out the centres of each of the 5 clusters as shown in Fig 4.8. Every cluster center is an array whose length is equal to 6 ,the number of features in our training dataset.

```
Cluster Centers:
[ 0.2755418   2.73065015 25.43343653 10.4654787   0.2879257   0.21671827]
[  0.73333333   1.          30.33333333 239.99193726   0.73333333
   0.53333333]
[  0.75         1.          31.89       132.95237487   0.64285714
   0.46428571]
[ 0.59259259  1.37037037 35.98148148 52.62068487  0.42592593  0.31481481]
[1.00000000e+00 1.00000000e+00 3.53333333e+01 5.12329224e+02
 3.33333333e-01 1.00000000e+00]
[ 0.31543624  2.0738255  46.03355705 19.98168321  0.27516779  0.19463087]
[ 0.56179775  2.56179775  8.74438202 26.98651588  0.49438202  0.2247191 ]
[ 0.68627451  1.09803922 36.39215686 81.40024537  0.54901961  0.50980392]
```

**Fig 4.8**

A sample of our Clustered result is displayed in Fig 4.9

| | Survived | PClass | Age | Fare | Gender | Boarded | features | prediction |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 3.0 | 22.0 | 7.250000 | 0.0 | 0.0 | [0.0, 3.0, 22.0, 7.25, 0.0, 0.0] | 0 |
| **1** | 1.0 | 1.0 | 38.0 | 71.283302 | 1.0 | 1.0 | [1.0, 1.0, 38.0, 71.2833023071289, 1.0, 1.0] | 7 |
| **2** | 1.0 | 3.0 | 26.0 | 7.925000 | 1.0 | 0.0 | [1.0, 3.0, 26.0, 7.925000190734863, 1.0, 0.0] | 0 |
| **3** | 1.0 | 1.0 | 35.0 | 53.099998 | 1.0 | 0.0 | [1.0, 1.0, 35.0, 53.099998474121094, 1.0, 0.0] | 3 |
| **4** | 0.0 | 3.0 | 35.0 | 8.050000 | 0.0 | 0.0 | [0.0, 3.0, 35.0, 8.050000190734863, 0.0, 0.0] | 0 |

**Fig 4.9**

We find the average of each of the following features and use these averages to see how the averages across the entire dataset stack up against average values of individual clusters.
We groupby prediction column which contains the clusters associated with every record and the total number of data points within each cluster is determined by count(). The result is given in Fig 5.0

| | prediction | avg(Survived) | avg(PClass) | avg(Age) | avg(Fare) | avg(Gender) | avg(Boarded) | count(prediction) |
|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0.275542 | 2.730650 | 25.433437 | 10.465479 | 0.287926 | 0.216718 | 323 |
| **1** | 1 | 0.733333 | 1.000000 | 30.333333 | 239.991937 | 0.733333 | 0.533333 | 15 |
| **2** | 2 | 0.750000 | 1.000000 | 31.890000 | 132.952375 | 0.642857 | 0.464286 | 28 |
| **3** | 3 | 0.592593 | 1.370370 | 35.981481 | 52.620685 | 0.425926 | 0.314815 | 54 |
| **4** | 4 | 1.000000 | 1.000000 | 35.333333 | 512.329224 | 0.333333 | 1.000000 | 3 |
| **5** | 5 | 0.315436 | 2.073826 | 46.033557 | 19.981683 | 0.275168 | 0.194631 | 149 |
| **6** | 6 | 0.561798 | 2.561798 | 8.744382 | 26.986516 | 0.494382 | 0.224719 | 89 |
| **7** | 7 | 0.686275 | 1.098039 | 36.392157 | 81.400245 | 0.549020 | 0.509804 | 51 |

**Fig 5.0**

From the data in Fig 5.0, we can infer the Survival rate for each of the clusters. For example, cluster 1 has a high survival rate and all first class passengers survived. There were mostly female survivors in this cluster. Similarly, information about the other clusters can be deduced.

The data for a single cluster is displayed in Fig 5.1.

| | Survived | PClass | Age | Fare | Gender | Boarded | features | prediction |
|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 1.0 | 19.0 | 263.000000 | 0.0 | 0.0 | [0.0, 1.0, 19.0, 263.0, 0.0, 0.0] | 1 |
| **1** | 1.0 | 1.0 | 23.0 | 263.000000 | 1.0 | 0.0 | [1.0, 1.0, 23.0, 263.0, 1.0, 0.0] | 1 |
| **2** | 0.0 | 1.0 | 24.0 | 247.520798 | 0.0 | 1.0 | [0.0, 1.0, 24.0, 247.5207977294922, 0.0, 1.0] | 1 |
| **3** | 1.0 | 1.0 | 50.0 | 247.520798 | 1.0 | 1.0 | [1.0, 1.0, 50.0, 247.5207977294922, 1.0, 1.0] | 1 |
| **4** | 1.0 | 1.0 | 18.0 | 262.375000 | 1.0 | 1.0 | [1.0, 1.0, 18.0, 262.375, 1.0, 1.0] | 1 |
| **5** | 1.0 | 1.0 | 24.0 | 263.000000 | 1.0 | 0.0 | [1.0, 1.0, 24.0, 263.0, 1.0, 0.0] | 1 |
| **6** | 0.0 | 1.0 | 27.0 | 211.500000 | 0.0 | 1.0 | [0.0, 1.0, 27.0, 211.5, 0.0, 1.0] | 1 |
| **7** | 1.0 | 1.0 | 42.0 | 227.524994 | 1.0 | 1.0 | [1.0, 1.0, 42.0, 227.52499389648438, 1.0, 1.0] | 1 |
| **8** | 0.0 | 1.0 | 64.0 | 263.000000 | 0.0 | 0.0 | [0.0, 1.0, 64.0, 263.0, 0.0, 0.0] | 1 |
| **9** | 1.0 | 1.0 | 15.0 | 211.337494 | 1.0 | 0.0 | [1.0, 1.0, 15.0, 211.33749389648438, 1.0, 0.0] | 1 |
| **10** | 1.0 | 1.0 | 18.0 | 227.524994 | 1.0 | 1.0 | [1.0, 1.0, 18.0, 227.52499389648438, 1.0, 1.0] | 1 |
| **11** | 1.0 | 1.0 | 38.0 | 227.524994 | 1.0 | 1.0 | [1.0, 1.0, 38.0, 227.52499389648438, 1.0, 1.0] | 1 |
| **12** | 1.0 | 1.0 | 29.0 | 211.337494 | 1.0 | 0.0 | [1.0, 1.0, 29.0, 211.33749389648438, 1.0, 0.0] | 1 |
| **13** | 1.0 | 1.0 | 21.0 | 262.375000 | 1.0 | 1.0 | [1.0, 1.0, 21.0, 262.375, 1.0, 1.0] | 1 |
| **14** | 1.0 | 1.0 | 43.0 | 211.337494 | 1.0 | 0.0 | [1.0, 1.0, 43.0, 211.33749389648438, 1.0, 0.0] | 1 |

**Fig 5.1**

## 4.4 Performance Evaluation

In this section, we describe the experimental results run on the setup described in the beginning of this section. We have extensively compared local execution of K-Means clustering algorithm using Scikit-learn library in Python with the clustering algorithm present in Spark MLLib library.First we checked for the optimal number of clusters to be generated. For this we used the Titanic data set, the number of clusters being taken is =8.
The results clearly showed that the performance of Spark turn out to be considerably higher in terms of time, where the dataset size results in a decrease in the processing time of up to twice times as compared to that of Python.
The results in the given tables. shows the difference in time taken for the execution in each case. Performance was also evaluated on the basis of memory usage. While the local implementation used a memory of 83.6KB whereas Spark used a memory of 16.9KB.

| **Dataset Size** | **Node** | **Time(ms)** | **Memory(KB)** |
|---|---|---|---|
| 2.6 MB | 1 | 204 | 16.9 |

**Table 2. Results for K-Means using Spark(MLlib)**

| **Dataset Size** | **Node** | **Time(ms)** | **Memory(Kb)** |
|---|---|---|---|
| 2.6 MB | 1 | 406 | 83.6 |

**Table 3. Results for K-Means on Python**

# CHAPTER 5. CONCLUSION

With the skyrocketing progress in the field of Big Data, processing becomes a crucial aspect of data analytics. Clustering is one of the most prominent method of data handling which in turn facilitates the prediction analysis on large datasets.

This paper provides a brief introduction into the world of Big Data Analysis followed by an overview of one of the most popular frameworks for handling big data, Apache Spark. It sketches two implementations of the K-Means Clustering Algorithm, one using the Spark framework and another without it. Moreover, performance analysis is done on both the implementations and a contrast is drawn. Our results for this analysis show that Spark is a very strong contender and would definitely bring about a change by using in-memory processing. Supporting multiple languages makes it versatile and user friendly. Observing Spark's ability to perform batch processing, streaming, and machine learning on the same cluster and looking at the current rate of adoption of Spark throughout the industry, Spark will be the de facto framework for a large number of use cases involving Big Data processing.
Despite having countless benefits, there are certain disadvantages of using Spark. One such limitations of Spark framework is that it is not computationally efficient for small data sets whereas Scikit Learn using Python is preferred for small datasets that fits the RAM. Furthermore, there is no file management system in Apache Spark. So, it depends upon other platforms like Hadoop or any other cloud-based platform for file management system. In addition, another limitation is that Apache Spark uses a significant amount of RAM to perform in-memory computation which is the reason that Spark is fast.

This report draws a parallel between the unsupervised learning on the synthetic dataset executed on Python and Apache Spark. Performance evaluation has been done between the two implementations and experimental results validate the efficiency of Spark framework for analysing large datasets.
For our future ventures, we aim at performing a comparative study of the execution performance of a clustering algorithm on a multi-cluster Spark setup as well as on a local machine. The multi-cluster Spark setup is expected reduce the time taken by a considerable amount and improve the accuracy of the algorithm by a huge margin. Thereafter, we aim at providing a contrast in the performance of both the cases in terms of time complexity and efficiency and thus establish the prowess of Apache Spark at present and the unique set of features it brings on the table.

We conclude that the performance of Spark turn out to be considerably higher in terms of time, where the dataset size results in a decrease in the processing time of up to twice times as compared to that in Python.

# REFERENCES


[1]A. Sinha and P. K. Jana, "A novel K-means based clustering algorithm for big data," *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Jaipur, 2016, pp. 1875-1879.

[2]P. P. Anchalia, A. K. Koundinya and S. N. K., "MapReduce Design of K-Means Clustering Algorithm," *2013 International Conference on Information Science and Applications (ICISA)*, Suwon, 2013, pp. 1-5.

[3] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, “Spark: Cluster computing with working sets.,” HotCloud, vol. 10, no. 10-10, p. 95, 2010.

[4] X. Meng, J. Bradley, B. Yavuz, E. Sparks, S. Venkataraman, D. Liu, J. Freeman, D. Tsai, M. Amde, S. Owen, et al., “Mllib: Machine learning in apache spark,” Journal of Machine Learning Research, vol. 17, no. 34, pp. 1–7, 2016

[5]Chung, S.S. and Aring, D. (2018) Integrated Real-Time Big Data Stream Sentiment Analysis Service. Journal of Data Analysis and Information Processing, 6, 46-66.

[6] Top 5 Apache Spark Use Cases(2016,Jun 16).[Online].Available: https://www.dezyre.com/article/top-5-apache-spark-usecases/271

[7] Andy Huang and Wei Wu(2014,Aug 14)[Online].Available: https://databricks.com/blog/2014/08/14/mining-graphdata-with-spark-at-alibaba-taobao.html

[8]Yurong Zhong and Dan Liu, "The application of K-means clustering algorithm based on Hadoop," *2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, Chengdu, 2016, pp. 88-92.

[9]Gopalani, Satish & Arora, Rohan. (2015). Comparing Apache Spark and Map Reduce with Performance Analysis using K-Means.*International Journal of Computer Applications (0975 – 8887)* Volume 113 – No. 1, March 2015

[10]Cui, Xiaoli and Zhu, Pingfei and Yang, Xin and Li, Keqiu and Ji, Changqing, Optimized big data K-means clustering using MapReduce, The Journal of Supercomputing, 70, 1249-1259 (2014)

[11]Laney, D. 3D data management: Controlling data volume, velocity and variety. META Group Research Note, 6, 70, (2001).

[12] Zhao, W., Ma, H., and He, Q. (2009). Parallel k-means clustering based on mapreduce. In Cloud Computing (pp. 674-679). Springer Berlin Heidelberg.

[13] Bahmani, B., Moseley, B., Vattani, A., Kumar, R., and Vassilvitskii, S.Scalable k-means ++ . Proceedings of the VLDB Endowment, 5(7), 622-633,(2012).

[14] Cai, X., Nie, F., and Huang, H. . Multi-view k-means clustering on big data. In Proceedings of the Twenty-Third international joint conference on Artificial Intelligence (pp. 2598-2604). AAAI Press.(2013)

[15]Spark: Cluster Computing with Working Sets Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, Ion Stoica University of California, Berkeley