A PROJECT ON NSE STOCK DATA ANALYSIS AND TESTING USING APACHE HADOOP

Project submitted and prepared by Bishal Biswas under guidance of Prof. Ashok Gupta

Project done as a part of assignments for
Post Graduate Diploma Program
From
Bombay Stock Exchange (BSE)
In collaboration with
Maulana Abul Kalam Azad University of Technology
(MAKAUT)

Certification of Approval

This document is hereby approved as credible study of the science subject carried out and represented in a manner to satisfy to the warrants of its acceptance as a prerequisite to the degree for which it has been submitted.

Moreover, it is understood that by this approval the undersigned does not necessarily endorse or approve any statements made, the opinion expressed or conclusion drawn therein but approved only for the sole purpose for which it has been indeed submitted.

Signatures of the Examiners with date.

×_______________________________

×_______________________________

×_______________________________

Dated:

Countersigned by:

×_______________________________

Prof. Ashok Gupta

Acknowledgement

Our project and everything started during the rule of SARS – CoVID19, virtually crippling the society and world as a whole sending everything into a lockdown, but still this course of Post Graduate Diploma in Data Science by BSE in collaboration with MAKAUT was made possible thanks to the diplomacy and steps taken by both institutes to combat the situation and make this course and project a possibility.

I want to take this opportunity of the project to thank the people at BSE and MAKAUT who provided us this opportunity to have an exposure to real life scenarios and the status of the present market. I also want to thank Prof. Ashok Gupta for guiding with every step from imparting knowledge about the subject to the intricacies of the HADOOP environment, clearing doubts and issues faced.

I am also grateful to my batchmates and peers where our collective knowledgebase and doubt clearing helped a lot in completing this project. Lastly, I want to thank my family for the mental support they provided me with in-spite of a loss.

×________________________

Bishal Biswas.
PGDDSPJULY2020/1
b.biswas_94587@ieee.org

Contents

Objective and Purpose

BIG Data is the next happening technology around the world and thought it is must for all Startups to keep abreast with the latest stuff to keep innovating and another reason is my fascination for BIG data that forces me to write about it. NO i am not suggesting that it is a new thing in India, in fact will give you some amazing instances where BIG Data helped to achieve BIG results.

To make it look simple let's just start with Analytics, we all know analytics is analysis of the data like how many and from where all people visited your website, for a small website the daily data may vary from 1GB to 50 GB. The more advanced for this kind of analysis can be sighted as CRM -Customer Relationship Management which stores more information about the people who are visiting your website.

World is growing fast with technology, access of World Wide Web became easier & number of internet users increasing day by day with help of new gadgets, PC, laptop, different devices and most importantly Smart Phones. With a revolution in digital media, social media networks, promotion, E Commerce, Smartphone Apps & Data, CC TV Camera & their Data Feed etc. a huge velocity of data is gathering worldwide and it's important to decipher the value & pattern of the data, store data information gathered & channelize it in a proper way. This is only possible with advance Big Data Technology.

Industries like IT, Retail, Manufacturing, Automobile, Financial Institute, E Commerce etc. are focusing in depth towards Big Data Concept because they have found out its importance, they know Data is Asset and its value will grow day by day and it can lead the Global business. Some benefits of it are:

- Data driven decision making with more accuracy.
- Customer active engagement.
- Operation optimization.
- Data driven Promotions.
- Preventing frauds & threats.
- Exploring new sources of revenue.
- Being ahead of your competitors.

The biggest challenge is to face and overcome in Big Data technology are Data encryption and information privacy.

As companies move to adopt new technologies, including big data analytics, some workers may be lost in the churn. It doesn't seem likely, based on how the job market has changed so far, that these new technologies will cause major shifts in employment rates, however — no matter how innovative or disruptive the tech is.

In the future, companies will still need employees to get work done. How that work is performed may change and the overall amount of work may decrease, but there's no evidence right now that suggests a coming collapse of the job market.

The growth of big data analytics will also probably be good for data scientists, especially those who have strong backgrounds in big data. Based on the growth of the big data analytics market in the past few years, along with the rising number of job openings, it's likely that demand for these skills will continue to increase in the near future.

Introduction

The art of uncovering the insights and trends in data has been around for centuries. The ancient Egyptians applied census data to increase efficiency in tax collection and they accurately predicted the flooding of the Nile river every year. Since then, people working in data science have carved out a unique and distinct field for the work they do.

IBM predicted that the demand for data scientists will increase by 28 percent by 2020. Another report indicates that in 2020, Data Science roles will expand to include machine learning (ML) and big data technology skills — especially given the rapid adoption of cloud and IoT technologies across global businesses.

In 2020, enterprises will demand more from their in-house data scientists, and these special experts will be viewed as "wizards of all business solutions." Another thing to note is that the annual demand for Data Science roles, which includes data engineers, data analysts, data developers and others, will hit the 700,000-mark next year.

To handle this behemoth of a need we need and with the changing job scenarios from predictive learning of a perspective recruits probability using their social network to automated systems like driverless cars to IoT connected devices in smart homes.

Big Data

The quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media can be aptly defined as data.

Big Data is also data but with a huge size. Big Data is a term used to describe a collection of data that is huge in volume and yet growing exponentially with time. In short, such data is so large and complex that none of the traditional data management tools are able to store it or process it efficiently.

Source of Big Data

Data typically originates from one of three primary sources of big data the internet/social networks, traditional business systems, and increasingly from the Internet of Things. The data from these sources can be structured, semi-structured, or unstructured, or any combination of these varieties.

Social Networks provide human-sourced information from:

- Twitter and Facebook
- Blogs and comments
- Pictures: Instagram®, Flickr™, Picasa™, etc.
- Videos: YouTube
- Internet searches
- Mobile data content (text messages)
- User-generated maps
- E-Mail

Traditional Business Systems like Visa, MasterCard, etc., these organizations offer customers services or products

- Commercial transactions
- Banking/stock records
- E-commerce
- Credit cards
- Medical records

Internet of Things data from

- Sensors: traffic, weather, mobile phone location, etc.
- Security, surveillance videos, and images
- Satellite images
- Data from computer systems (logs, web logs, etc.)
- Samsung LYNX devices.
- Smart lighting systems by Wipro, Philips, Legrand, etc.
- ESP – 8266 based devices
- Devices connected to Thingspeak.com
- Google Home, Amazon Echo, Samsung Bixby, etc., with connected systems

Types of Data

Big Data could be found in three forms:

- Structured
- Unstructured
- Semi-structured

Structured

Any data that can be stored, accessed and processed in the form of fixed format is termed as a 'structured' data. Over the period of time, talent in computer science has achieved greater success in developing techniques for working with such kind of data (where the format is well known in advance) and also deriving value out of it. However, nowadays, we are foreseeing issues when a size of such data grows to a huge extent, typical sizes are being in the rage of multiple zettabytes.

Unstructured

Any data with unknown form or the structure is classified as unstructured data. In addition to the size being huge, un-structured data poses multiple challenges in terms of its processing for deriving value out of it. A typical example of unstructured data is a heterogeneous data source containing a combination of simple text files, images, videos etc. Now day organizations have wealth of data available with them but unfortunately, they don't know how to derive value out of it since this data is in its raw form or unstructured format.

Semi-structured

Semi-structured data can contain both the forms of data. We can see semi-structured data as a structured in form but it is actually not defined with e.g. a table definition in relational DBMS.

Characteristics of Big Data

(i) Volume – The name Big Data itself is related to a size which is enormous. Size of data plays a very crucial role in determining value out of data. Also, whether a particular data can actually be considered as a Big Data or not, is dependent upon the volume of data. Hence, 'Volume' is one characteristic which needs to be considered while dealing with Big Data.

(ii) Variety – Variety refers to heterogeneous sources and the nature of data, both structured and unstructured. During earlier days, spreadsheets and databases were the only sources of data considered by most of the applications. Nowadays, data in the form of emails, photos, videos, monitoring devices, PDFs, audio, etc. are also being considered in the analysis applications. This variety of unstructured data poses certain issues for storage, mining and analysing data.

(iii) Velocity – The term 'velocity' refers to the speed of generation of data. How fast the data is generated and processed to meet the demands, determines real potential in the data. Big Data Velocity deals with the speed at which data flows in from sources like business processes, application logs, networks, and social media sites, sensors, Mobile devices, etc. The flow of data is massive and continuous.

(iv) Variability – This refers to the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.

Handling Big Data

Developers prefer to avoid vendor lock-in and tend to use free tools for the sake of versatility, as well as due to the possibility to contribute to the evolvement of their beloved platform. Open source products boast the same, if not better level of documentation depth, along with a much more dedicated support from the community, who are also the product developers and Big Data practitioners, who know what they need from a product. Thus said, this is the list of 8 hot Big Data tool to use in 2018, based on popularity, feature richness and usefulness.

1. Apache Hadoop

The long-standing champion in the field of Big Data processing, well-known for its capabilities for huge-scale data processing. This open source Big Data framework can run on-prem or in the cloud and has quite low hardware requirements. The main Hadoop benefits and features are as follows:

HDFS — Hadoop Distributed File System, oriented at working with huge-scale bandwidth

MapReduce — a highly configurable model for Big Data processing

YARN — a resource scheduler for Hadoop resource management

Hadoop Libraries — the needed glue for enabling third party modules to work with Hadoop

2. Apache Spark

Apache Spark is the alternative — and in many aspects the successor — of Apache Hadoop. Spark was built to address the shortcomings of Hadoop and it does this incredibly well. For example, it can process both batch data and real-time data, and operates 100 times faster than MapReduce. Spark provides the in-memory data processing capabilities, which is way faster than disk processing leveraged by MapReduce. In addition, Spark works with HDFS, OpenStack and Apache Cassandra, both in the cloud and on-prem, adding another layer of versatility to big data operations for your business.

3. Apache Storm

Storm is another Apache product, a real-time framework for data stream processing, which supports any programming language. Storm scheduler balances the workload between multiple nodes based on topology configuration and works well with Hadoop HDFS. Apache Storm has the following benefits:

- Great horizontal scalability
- Built-in fault-tolerance
- Auto-restart on crashes
- Clojure-written
- Works with Direct Acyclic Graph (DAG) topology
- Output files are in JSON format

4. Apache Cassandra

Apache Cassandra is one of the pillars behind Facebook’s massive success, as it allows to process structured data sets distributed across huge number of nodes across the globe. It works well under heavy workloads due to its architecture without single points of failure and boasts unique capabilities no other NoSQL or relational DB has, such as:

Great liner scalability
Simplicity of operations due to a simple query language used
Constant replication across nodes
Simple adding and removal of nodes from a running cluster
High fault tolerance
Built-in high-availability

5. MongoDB

MongoDB is another great example of an open source NoSQL database with rich features, which is cross-platform compatible with many programming languages. IT Svit uses MongoDB in a variety of cloud computing and monitoring solutions, and we specifically developed a module for automated MongoDB backups using Terraform. The most prominent MongoDB features are:

Stores any type of data, from text and integer to strings, arrays, dates and boolean
Cloud-native deployment and great flexibility of configuration
Data partitioning across multiple nodes and data centres
Significant cost savings, as dynamic schemas enable data processing on the go

6. R Programming Environment

R is mostly used along with JuPyteR stack (Julia, Python, R) for enabling wide-scale statistical analysis and data visualization. JupyteR Notebook is one of 4 most popular Big Data visualization tools, as it allows composing literally any analytical model from more than 9,000 CRAN (Comprehensive R Archive Network) algorithms and modules, running it in a convenient environment, adjusting it on the go and inspecting the analysis results at once. The main benefits of using R are as follows:

R can run inside the SQL server
R runs on both Windows and Linux servers
R supports Apache Hadoop and Spark
R is highly portable
R easily scales from a single test machine to vast Hadoop data lakes

7. Neo4j

Neo4j is an open source graph database with interconnected node-relationship of data, which follows the key-value pattern in storing data. IT Svit has recently built a resilient AWS infrastructure with Neo4j for one of our customers and the database performs well under heavy workload of network data and graph-related requests. Main Neo4j features are as follows:

Built-in support for ACID transactions
Cypher graph query language
High-availability and scalability
Flexibility due to the absence of schemas
Integration with other databases

8. Apache SAMOA

This is another of the Apache family of tools used for Big Data processing. Samoa specializes at building distributed streaming algorithms for successful Big Data mining. This tool is built with pluggable architecture and must be used atop other Apache products like Apache Storm we mentioned earlier. Its other features used for Machine Learning include the following:

Clustering
Classification
Normalization
Regression
Programming primitives for building custom algorithms

Using Apache Samoa enables the distributed stream processing engines to provide such tangible benefits:

Program once, use anywhere
Reuse the existing infrastructure for new projects
No reboot or deployment downtime
No need for backups or time-consuming updates

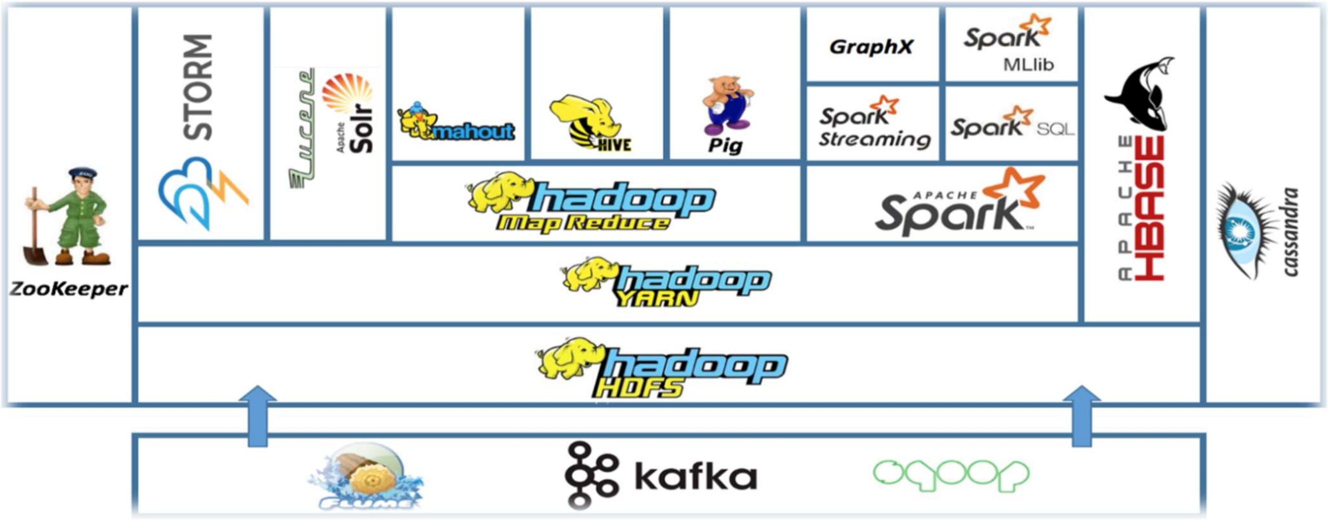
Why Hadoop?

The Apache™ Hadoop® project develops open-source software for reliable, scalable, distributed computing. The Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models. It is designed to scale up from single servers to thousands of machines, each offering local computation and storage. Rather than rely on hardware to deliver high-availability, the library itself is designed to detect and handle failures at the application layer, so delivering a highly-available service on top of a cluster of computers, each of which may be prone to failures.

Modules of Apache™ Hadoop®

The project includes these modules:

- Hadoop Common: The common utilities that support the other Hadoop modules.
- Hadoop Distributed File System (HDFS™): A distributed file system that provides high-throughput access to application data.
- Hadoop YARN: A framework for job scheduling and cluster resource management.
- Hadoop MapReduce: A YARN-based system for parallel processing of large data sets.
- Hadoop Ozone: An object store for Hadoop



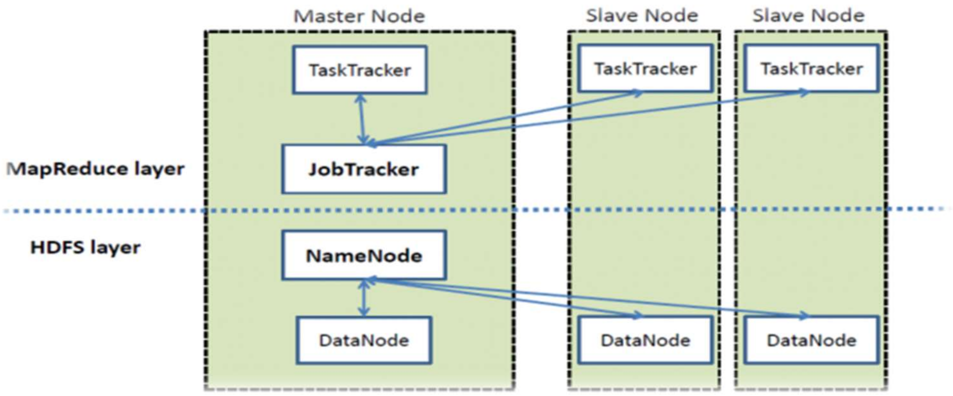
All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework. Apache Hadoop's MapReduce and HDFS components originally derived respectively from Google's MapReduce and Google File System (GFS) papers.

Beyond HDFS, YARN and MapReduce, the entire Apache Hadoop "platform" is now commonly considered to consist of a number of related projects as well: Apache Pig, Apache Hive, Apache HBase, and others. For the end-users, though MapReduce Java code is common, any programming language can be used with "Hadoop Streaming" to implement the "map" and "reduce" parts of the user's program. Apache Pig and Apache Hive, among other related projects, expose higher level user interfaces like Pig latin and a SQL variant respectively. The Hadoop framework itself is mostly written in the Java programming language, with some native code in C and command line utilities written as shell-scripts.

HDFS and MapReduce

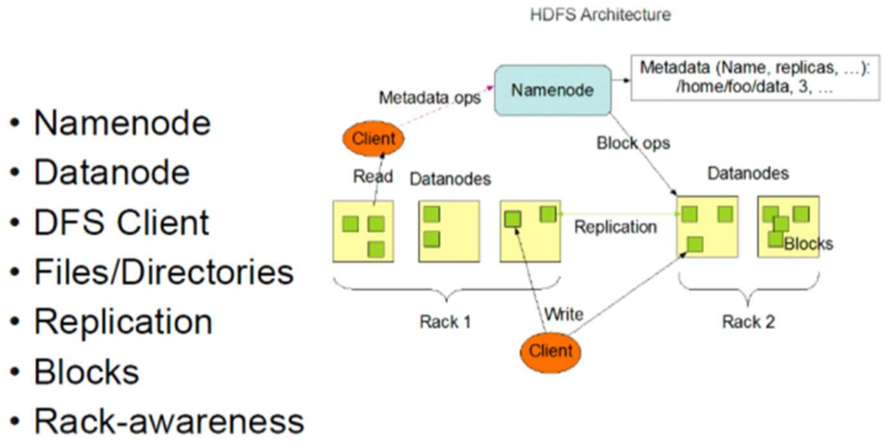
There are two primary components at the core of Apache Hadoop 1.x: the Hadoop Distributed File System (HDFS) and the MapReduce parallel processing framework. These are both open source projects, inspired by technologies created inside Google.

Hadoop distributed file system



Hadoop distributed file system

The Hadoop distributed file system (HDFS) is a distributed, scalable, and portable file-system written in Java for the Hadoop framework. Each node in a Hadoop instance typically has a single namenode, and a cluster of datanodes form the HDFS cluster. The situation is typical because each node does not require a datanode to be present. Each datanode serves up blocks of data over the network using a block protocol specific to HDFS. The file system uses the TCP/IP layer for communication. Clients use Remote procedure call (RPC) to communicate between each other.



- Namenode
- Datanode
- DFS Client
- Files/Directories
- Replication
- Blocks
- Rack-awareness

HDFS stores large files (typically in the range of gigabytes to terabytes) across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence does not require RAID storage on hosts. With the default replication value, 3, data is stored on three nodes: two on the same rack, and one on a different rack. Data nodes can talk to each other to rebalance data, to move copies around, and to keep the replication of data high. HDFS is not fully POSIX-compliant, because the requirements for a POSIX file-system differ from the target goals for a Hadoop application. The trade-off of not having a fully POSIX-compliant file-system is increased performance for data throughput and support for non-POSIX operations such as Append.

HDFS added the high-availability capabilities for release 2.x, allowing the main metadata server (the NameNode) to be failed over manually to a backup in the event of failure, automatic fail-over.

The HDFS file system includes a so-called secondary namenode, which misleads some people into thinking that when the primary namenode goes offline, the secondary namenode takes over. In fact, the secondary namenode regularly connects with the primary namenode and builds snapshots of the primary namenode's directory information, which the system then saves to local or remote directories. These checkpointed images can be used to restart a failed primary namenode without having to replay the entire journal of file-system actions, then to edit the log to create an up-to-date directory structure. Because the namenode is the single point for storage and management of metadata, it can become a bottleneck for supporting a huge number of files, especially a large number of small files. HDFS Federation, a new addition, aims to tackle this problem to a certain extent by allowing multiple name-spaces served by separate namenodes.

An advantage of using HDFS is data awareness between the job tracker and task tracker. The job tracker schedules map or reduce jobs to task trackers with an awareness of the data location. For example, if node A contains data (x, y, z) and node B contains data (a, b, c), the job tracker schedules node B to perform map or reduce tasks on (a,b,c) and node A would be scheduled to perform map or reduce tasks on (x,y,z). This reduces the amount of traffic that goes over the network and prevents unnecessary data transfer. When Hadoop is used with other file systems, this advantage is not always available. This can have a significant impact on job-completion times, which has been demonstrated when running data-intensive jobs. HDFS was designed for mostly immutable files and may not be suitable for systems requiring concurrent write-operations.

Another limitation of HDFS is that it cannot be mounted directly by an existing operating system. Getting data into and out of the HDFS file system, an action that often needs to be performed before and after executing a job, can be inconvenient. A filesystem in Userspace (FUSE) virtual file system has been developed to address this problem, at least for Linux and some other Unix systems.

File access can be achieved through the native Java API, the Thrift API, to generate a client in the language of the users' choosing (C++, Java, Python, PHP, Ruby, Erlang, Perl, Haskell, C#, Cocoa, Smalltalk, or OCaml), the command-line interface, or browsed through the HDFS-UI web app over HTTP.

YARN enhances the power of a Hadoop compute cluster in the following ways:

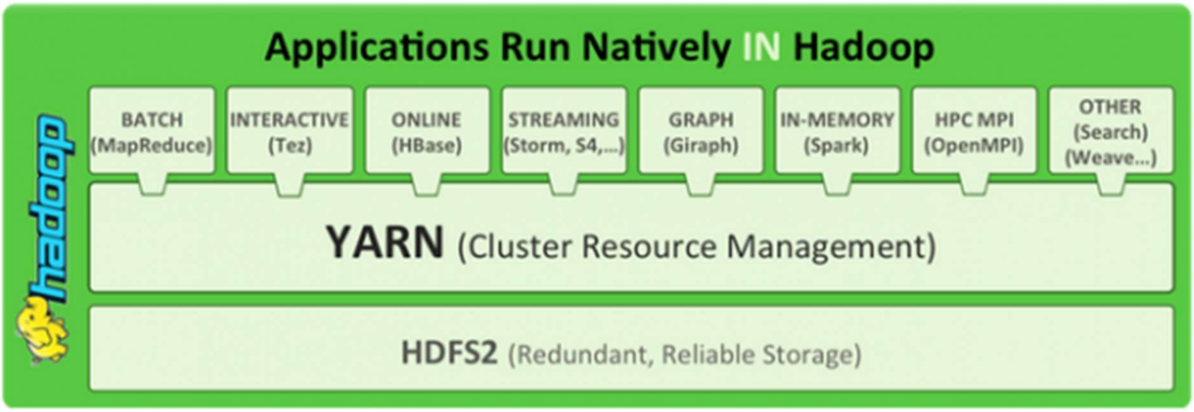
Scalability: The processing power in data centers continues to grow quickly. Because YARN ResourceManager focuses exclusively on scheduling, it can manage those larger clusters much more easily.

Compatibility with MapReduce: Existing MapReduce applications and users can run on top of YARN without disruption to their existing processes.

Improved cluster utilization: The ResourceManager is a pure scheduler that optimizes cluster utilization according to criteria such as capacity guarantees, fairness, and SLAs. Also, unlike before, there are no named map and reduce slots, which helps to better utilize cluster resources.

Support for workloads other than MapReduce: Additional programming models such as graph processing and iterative modeling are now possible for data processing. These added models allow enterprises to realize near real-time processing and increased ROI on their Hadoop investments.

Agility: With MapReduce becoming a user-land library, it can evolve independently of the underlying resource manager layer and in a much more agile manner.


Applications Run Natively IN Hadoop
BATCH (MapReduce)
INTERACTIVE (Tez)
ONLINE (HBase)
STREAMING (Storm, S4,...)
GRAPH (Giraph)
IN-MEMORY (Spark)
HPC MPI (OpenMPI)
OTHER (Search) (Weave...)
YARN (Cluster Resource Management)
HDFS2 (Redundant, Reliable Storage)
hadoop

How YARN works

The fundamental idea of YARN is to split up the two major responsibilities of the JobTracker/TaskTracker into separate entities:

- a global ResourceManager
- a per-application ApplicationMaster
- a per-node slave NodeManager and
- a per-application container running on a NodeManager

The ResourceManager and the NodeManager form the new, and generic, system for managing applications in a distributed manner. The ResourceManager is the ultimate authority that arbitrates resources among all the applications in the system. The per-application ApplicationMaster is a framework-specific entity and is tasked with negotiating resources from the ResourceManager and working with the NodeManager(s) to execute and monitor the component tasks. The ResourceManager has a scheduler, which is responsible for allocating resources to the various running applications, according to constraints such as queue capacities, user-limits etc. The scheduler performs its scheduling function based on the resource requirements of the applications. The NodeManager is the per-machine slave, which is responsible for launching the applications' containers, monitoring their resource usage (cpu, memory, disk, network) and reporting the same to the

ResourceManager. Each ApplicationMaster has the responsibility of negotiating appropriate resource containers from the scheduler, tracking their status, and monitoring their progress. From the system perspective, the ApplicationMaster runs as a normal container.

About Apache™ MapReduce®

Hadoop MapReduce is a software framework for easily writing applications which process vast amounts of data (multi-terabyte data-sets) in-parallel on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner.

A MapReduce job usually splits the input data-set into independent chunks which are processed by the map tasks in a completely parallel manner. The framework sorts the outputs of the maps, which are then input to the reduce tasks. Typically, both the input and the output of the job are stored in a file-system. The framework takes care of scheduling tasks, monitoring them and re-executes the failed tasks.

Typically the compute nodes and the storage nodes are the same, that is, the MapReduce framework and the Hadoop Distributed File System (see HDFS Architecture Guide) are running on the same set of nodes. This configuration allows the framework to effectively schedule tasks on the nodes where data is already present, resulting in very high aggregate bandwidth across the cluster.

The MapReduce framework consists of a single master JobTracker and one slave TaskTracker per cluster-node. The master is responsible for scheduling the jobs' component tasks on the slaves, monitoring them and re-executing the failed tasks. The slaves execute the tasks as directed by the master.

Minimally, applications specify the input/output locations and supply map and reduce functions via implementations of appropriate interfaces and/or abstract-classes. These, and other job parameters, comprise the job configuration. The Hadoop job client then submits the job (jar/executable etc.) and configuration to the JobTracker which then assumes the responsibility of distributing the software/configuration to the slaves, scheduling tasks and monitoring them, providing status and diagnostic information to the job-client.

Although the Hadoop framework is implemented in JavaTM, MapReduce applications need not be written in Java.

- Hadoop Streaming is a utility which allows users to create and run jobs with any executables (e.g. shell utilities) as the mapper and/or the reducer.
- Hadoop Pipes is a SWIG- compatible C++ API to implement MapReduce applications (non JNITM based).

Hadoop streaming is a utility that comes with the Hadoop distribution. The utility allows you to create and run Map/Reduce jobs with any executable or script as the mapper and/or the reducer. For example:

```
mapred streaming \
  -input myInputDirs \
  -output myOutputDir \
  -mapper /bin/cat \
  -reducer /usr/bin/wc
```

Working procedure of Streaming: In the above example, both the mapper and the reducer are executables that read the input from stdin (line by line) and emit the output to stdout. The utility will create a Map/Reduce job, submit the job to an appropriate cluster, and monitor the progress of the job until it completes.

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

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

This is the basis for the communication protocol between the Map/Reduce framework and the streaming mapper/reducer.

About Apache™ Hive®

The Apache Hive ™ data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL. Structure can be projected onto data already in storage. A command line tool and JDBC driver are provided to connect users to Hive.

About Apache™ Pig®

Apache Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs, coupled with infrastructure for evaluating these programs. The salient property of Pig programs is that their structure is amenable to substantial parallelization, which in turns enables them to handle very large data sets.

At the present time, Pig's infrastructure layer consists of a compiler that produces sequences of Map-Reduce programs, for which large-scale parallel implementations already exist (e.g., the Hadoop subproject). Pig's language layer currently consists of a textual language called Pig Latin, which has the following key properties:

- Ease of programming. It is trivial to achieve parallel execution of simple, "embarrassingly parallel" data analysis tasks. Complex tasks comprised of multiple interrelated data transformations are explicitly encoded as data flow sequences, making them easy to write, understand, and maintain.
- Optimization opportunities. The way in which tasks are encoded permits the system to optimize their execution automatically, allowing the user to focus on semantics rather than efficiency.
- Extensibility. Users can create their own functions to do special-purpose processing.

About Apache™ HBASE™

Apache HBase™ is the Hadoop database, a distributed, scalable, big data store. Use of Apache HBase™ comes when you need random, realtime read/write access to your Big Data. This project's goal is the hosting of very large tables -- billions of rows X millions of columns -- atop clusters of commodity hardware. Apache HBase is an open-source, distributed, versioned, non-relational database modeled after Google's Bigtable: A Distributed Storage System for Structured Data by Chang et al. Just as Bigtable leverages the distributed data storage provided by the Google File System, Apache HBase provides Bigtable-like capabilities on top of Hadoop and HDFS.

Features of Hbase™

- Linear and modular scalability.
- Strictly consistent reads and writes.
- Automatic and configurable sharding of tables
- Automatic failover support between RegionServers.
- Convenient base classes for backing Hadoop MapReduce jobs with Apache HBase tables.
- Easy to use Java API for client access.
- Block cache and Bloom Filters for real-time queries.
- Query predicate push down via server-side Filters

- Thrift gateway and a REST-ful Web service that supports XML, Protobuf, and binary data encoding options
- Extensible jruby-based (JIRB) shell
- Support for exporting metrics via the Hadoop metrics subsystem to files or Ganglia; or via JMX

About Apache™ Spark®

Apache Spark is a unified analytics engine for large-scale data processing. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs. It also supports a rich set of higher-level tools including Spark SQL for SQL and structured data processing, MLlib for machine learning, GraphX for graph processing, and Structured Streaming for incremental computation and stream processing.

A Comparative Study of the APIs

1. Pig:

Pig is used for the analysis of a large amount of data. It is abstract over MapReduce. Pig is used to perform all kinds of data manipulation operations in Hadoop. It provides the Pig-Latin language to write the code that contains many inbuilt functions like join, filter, etc. The two parts of the Apache Pig are Pig-Latin and Pig-Engine. Pig Engine is used to convert all these scripts into a specific map and reduce tasks. Pig abstraction is at a higher level. It contains less line of code as compared to MapReduce.

2. Hive:

Hive is built on the top of Hadoop and is used to process structured data in Hadoop. Hive was developed by Facebook. It provides various types of querying language which is frequently known as Hive Query Language. Apache Hive is a data warehouse and which provides an SQL-like interface between the user and the Hadoop distributed file system (HDFS) which integrates Hadoop.

Difference between Pig and Hive:

S.NO.	PIG	HIVE
1.	Pig operates on the client side of a cluster.	Hive operates on the server side of a cluster.
2.	Pig uses pig-latin language.	Hive uses HiveQL language.
3.	Pig is a Procedural Data Flow Language.	Hive is a Declarative SQLish Language.
4.	It was developed by Yahoo.	It was developed by Facebook.

5.	It is used by Researchers and Programmers.	It is mainly used by Data Analysts.
6.	It is used to handle structured and semi-structured data.	It is mainly used to handle structured data.
7.	It is used for programming.	It is used for creating reports.
8.	Pig scripts end with '.pig' extension.	In Hive, all extensions are supported.
9.	It does not support partitioning.	It supports partitioning.
10.	It loads data quickly.	It loads data slowly.
11.	It does not support JDBC.	It supports JDBC.
12.	It does not support ODBC.	It supports ODBC.
13.	Pig does not have a dedicated metadata database.	Hive makes use of the exact variation of dedicated SQL-DDL language by defining tables beforehand.
14.	It supports Avro file format.	It does not support Avro file format.
15.	Pig is suitable for complex and nested data structures.	Hive is suitable for batch-processing OLAP systems.
16.	Pig does not support schema to store data.	Hive supports schema for data insertion in tables.

Hive:
Hive is a data-warehousing package built on the top of Hadoop. It is mainly used for data analysis. It generally targets towards users already comfortable with Structured Query Language (SQL). It is very similar to SQL and called Hive Query Language (HQL). Hive manages and queries structured data. Moreover, hive abstracts complexity of Hadoop. Hive was developed by Facebook in 2007 to handle massive amount of data. It does not support:

- Not a full database.
- Not a real time processing system.
- Not SQL-92 compliant.
- Does not provide row level insert, updates or deletes.
- Doesn't support transactions and limited sub-query support.
- Query optimization in evolving stage.

HBase:
HBase is a column-oriented database management system that runs on top of Hadoop Distributed File System (HDFS). It is well suited for sparse data sets, which are common in many big data use cases. It is an opensource, distributed database developed by Apache software foundations. Initially, it was named Google Big Table, afterwards it was re-named as HBase and is primarily written in Java. It can store massive amount of data from terabytes to petabytes. It is built for

low-latency operations and is used extensively for read and write operations. It stores large amount of data in the form of tables.

Difference between Hive and HBase:

HIVE	HBASE
Hive is a query engine	Data storage particularly for unstructured data
Mainly used for batch processing	Extensively used for transactional processing
Not a real time processing	Real-time processing
Only for analytical queries	Real-time querying
Runs on the top of Hadoop	Runs on the top of HDFS (Hadoop distributed file system)
Apache Hive is not a database	It supports NoSQL database
It has schema model	It is free from schema model
Made for high latency operations	Made for low level latency operations

Relational Database Management System (RDBMS) –

RDBMS is for SQL, and for all modern database systems like MS SQL Server, IBM DB2, Oracle, MySQL, and Microsoft Access. A Relational database management system (RDBMS) is a database management system (DBMS) that is based on the relational model as introduced by E. F. Codd. An RDBMS is a type of DBMS with a row-based table structure that connects related data elements and includes functions that maintain the security, accuracy, integrity and consistency of the data. The most basic RDBMS functions are create, read, update and delete operations. HBase follows the ACID Properties.

HBase –

HBase is a column-oriented database management system that runs on top of Hadoop Distributed File System (HDFS). It is well suited for sparse data sets, which are common in many big data use cases. It is an opensource, distributed database developed by Apache software foundations. Initially, it was named Google Big Table, afterwards it was re-named as HBase and is primarily written in Java. It can store massive amount of data from terabytes to petabytes. It is built for low-latency operations and is used extensively for read and write operations. It stores large amount of data in the form of tables.

Difference between RDBMS and HBase:

RDBMS	HBASE
It requires SQL (structured query language)	NO SQL
It has a fixed schema	No fixed schema
It is row oriented	It is column oriented
It is not scalable	It is scalable
It is static in nature	Dynamic in nature

Slower retrieval of data	Faster retrieval of data
It follows the ACID (Atomicity, Consistency, Isolation and Durability) property.	It follows CAP (Consistency, Availability, Partition-tolerance) theorem.
It can handle structured data	It can handle structured, unstructured as well as semi-structured data
It cannot handle sparse data	It can handle sparse data

Hadoop: Hadoop is a Framework or Software which was invented to manage huge data or Big Data. Hadoop is used for storing and processing large data distributed across a cluster of commodity servers. Hadoop stores the data using Hadoop distributed file system and process/query it using the Map-Reduce programming model.

Hive: Hive is an application that runs over the Hadoop framework and provides SQL like interface for processing/query the data. Hive is designed and developed by Facebook before becoming part of the Apache-Hadoop project. Hive runs its query using HQL (Hive query language). Hive is having the same structure as RDBMS and almost the same commands can be used in Hive. Hive can store the data in external tables so it's not mandatory to used HDFS also it supports file formats such as ORC, Avro files, Sequence File and Text files, etc.

Differences between Hadoop and Hive:

HADOOP	**HIVE**
Hadoop is a framework to process/query the Big data	**Hive** is an SQL Based tool that builds over Hadoop to process the data.
Hadoop can understand Map Reduce only.	**Hive** process/query all the data using HQL (Hive Query Language) it's SQL-Like Language
Map Reduce is an integral part of **Hadoop**	**Hive's** query first get converted into Map Reduce than processed by Hadoop to query the data.
Hadoop understands SQL using Java-based Map Reduce only.	**Hive** works on SQL Like query
In **Hadoop**, have to write complex Map Reduce programs using Java which is not similar to traditional Java.	In **Hive**, earlier used traditional "Relational Database's" commands can also be used to query the big data
Hadoop is meant for all types of data whether it is Structured, Unstructured or Semi-Structured.	**Hive** can only process/query the structured data

In the simple **Hadoop** ecosystem, the need to write complex Java programs for the same data.	Using **Hive**, one can process/query the data without complex programming
One side **Hadoop** frameworks need 100s line for preparing Java-based MR program	**Hive** can query the same data using 8 to 10 lines of HQL.

Tasks required and data provided

Analysis tasks:

1. Use the given csv file as input data and implement following transformations: Filter Rows on specified criteria "Symbol equals GEOMETRIC" Select specific columns from those available: SYMBOL, OPEN, HIGH, LOW and CLOSE which meets above criteria Generate count of the number of rows from above result
2. Calculation of various statistical quantities and decision making: Only lines with value "EQ" in the "series" column should be processed. As the first stage, filter out all the lines that do not fulfil this criteria. For every stock(with value "EQ" in the "series"), for every year, calculate the statistical parameters(Minimum, Maximum, Mean and Standard Deviation) and store the generated information in properly designated tables.
3. Select any year for which data is available: For the selected year, create a table that contains data only for those stocks that have an average total traded quntity of 3 lakhs or more per day. Print out the first 25 entries of the table and submit. From above output, select any 10 stocks from IT ('HCLTECH', 'NIITTECH', 'TATAELXSI','TCS', 'INFY', 'WIPRO', 'DATAMATICS','TECHM','MINDTREE' and 'OFSS') and create a table combining their data. Find out the Pearsons Correlation Coeffecient for every pair of stocks you have selected above. Final output should be in decreasing order of the coeffecient.
4. Use the coorrelation information generated in step 3 in the following way: a. Assume you have Rs10 lakh to invest b. Assume you have to invest in six stocks on the first working day of January of the next year. c. By using logic/simulation/etc. Identify the stocks that you will invest in, such that at the end of the year: At least your overall capital (Rs 10 lakh) is protected.

Data definitions:

- SYMBOL string – company name
- SERIES string – company series type
- OPEN float – opening price

- HIGH float – highest of day
- LOW float – lowest of day
- CLOSE float – closing price
- LAST float – last working day's best price
- PREVCLOSE float – previous closing price
- TOTTRDQTY int – total traded quantity
- TOTTRDVAL float – total traded value
- TIMESTAMPS string – timestamp at a point of linear time series of stock data
- TOTALTRADES int – total trades of a company
- ISIN string – unique digital signature of the transacted data of stocks recorded

Pre-information: data is a randomised timeseries data of NSE stock values over a period of 2 years from 2016 to 2017 where end limits of both years are satisfied

Data obtained from: https://www.kaggle.com/minatverma/nse-stocks-data?select=FINAL_FROM_DF.csv

Codes and Workflows

Question 1

Hive Codes

```
-- Analysis 1:
--  Use the given csv file as input data and implement following transformations:
-- a. Filter Rows on specified criteria "Symbol equals GEOMETRIC"
-- b. Select specific columns from those available: SYMBOL, OPEN, HIGH, LOW and CLOSE which meets above criteria
-- c. Generate count of the number of rows from above result

-- start of codes
-- one time jobs
-- **## Please avoid lines here on forward if nsestocksdb is available in 'show databases' command and contains data                ##**
-- ** and jump to the section of analysis **
create database nsestocksdb;
use nsestocksdb;
--create dataset
create table data_raw_headless
(SYMBOL string,SERIES string,OPEN float,HIGH float,LOW float,CLOSE float,LAST float,PREVCLOSE float,
TOTTRDQTY int,TOTTRDVAL float,TIMESTAMPS string,TOTALTRADES int,ISIN string)
row format delimited fields terminated by ',' lines terminated by '\n'
tblproperties("skip.header.line.count"="1");
```

```
load data inpath 'hdfs://localhost:9000/user/hive/warehouse/FINAL_FROM_DF.csv' into table
data_raw_headless;
-- find number of datapoints
select count(*) from data_raw_headless where isin!='';
-- result 846404

-- analysis jobs
--
create table anlysjob1a as select * from data_raw_headless where symbol=='GEOMETRIC';
create table anlysjob1b as select symbol,open,high,low,close from anlysjob1a;
select count(*) from anlysjob1b



-- obtained result
--hive> select count(*) from anlysjob1b ;--
--OK
--295
--Time taken: 0.441 seconds, Fetched: 1 row(s)
--hive>
```

Pig Codes

```
/*Analysis 1:
 Use the given csv file as input data and implement following transformations:
 a. Filter Rows on specified criteria "Symbol equals GEOMETRIC"
 b. Select specific columns from those available: SYMBOL, OPEN, HIGH, LOW and CLOSE which meets
above criteria
 c. Generate count of the number of rows from above result */

-- start of code

--raw data load with defined schema
data_raw= LOAD '/home/kali/Hadoop/Local_Datasets/FINAL_FROM_DF.csv' USING PigStorage(',') as
(SYMBOL:chararray,SERIES:chararray,OPEN:float,HIGH:float,LOW:float,CLOSE:float,LAST:float,PREVCLOSE:fl
oat,TOTTRDQTY:int,TOTTRDVAL:float,TIMESTAMPS:Datetime,TOTALTRADES:int,ISIN:chararray);
data_collect = FILTER data_raw BY (OPEN>=0); --cleaning with condition (collect = headless)
rawGrp = GROUP data_raw ALL; --checker group
cltGrp = GROUP data_collect ALL; --checker group
ctrrw = FOREACH rawGrp GENERATE COUNT(data_raw);  -- counter value (846405)
ctrclt = FOREACH clnGrp GENERATE COUNT(data_collect); -- counter value (846404)
dump ctrrw --trigger calculation
dump ctrclt --trigger calculation

--no visible debris ... collect-> headless
--
--
```

```
--analysis job
anlysjob1a = FILTER data_collect BY (SYMBOL=='GEOMETRIC');  -- option -a
anlysjob1b = FOREACH anlysjob1a GENERATE SYMBOL,OPEN,HIGH,LOW,CLOSE ;  --option -b
soln_grp = GROUP anlysjob1b ALL; -- group counter
anlysjob1c = FOREACH soln_grp GENERATE COUNT(anlysjob1b) ;

-- Storage and dump
STORE anlysjob1c INTO '/home/kali/Hadoop/Results/pig_results2/analysis1/' USING PigStorage();
dump anlysjob1c;

/*
dump value: 295
stored
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results2/analysis1/part*
295
kali@kali:~$
*/
```

Spark Codes

```
// Analysis 1:
//  Use the given csv file as input data and implement following transformations:
// a. Filter Rows on specified criteria "Symbol equals GEOMETRIC"
// b. Select specific columns from those available: SYMBOL, OPEN, HIGH, LOW and CLOSE which meets
above criteria
// c. Generate count of the number of rows from above result



//Start of Code
//cleaning of visible debris
//import
var data_raw= sc.textFile("hdfs://localhost:9000/assign2/FINAL_FROM_DF.csv")
var data_split=data_raw.map(x=>x.split(',')) //split by csv terms
var data_headless=data_split.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1) else iter }
//removing header line
data_headless.count // count confirm by 846404

//visible debris not noted ->  no additional filters imposed
//Data Scheme: SYMBOL    SERIES    OPEN     HIGH     LOW      CLOSE    LAST      PREVCLOSE
             TOTTRDQTY        TOTTRDVAL        TIMESTAMP        TOTALTRADES      ISIN


//analysis job
//
var analysisjob1a=data_headless.filter(x=>{x(0) == "GEOMETRIC"})  // option a
var analysisjob1b=analysisjob1a.map(x=>(x(0),x(2),x(3),x(4),x(5)))  // option b
var analysisjob1c=analysisjob1b.count // option c
```

```
// end of code

//Result obtained :
//***
//.
//scala> analysisjob1c
//res87: Long = 295
//
//scala>
//
//.*****
```

Question 2

Hive Codes

```
-- Analysis 2:
--   Calculation of various statistical quantities and decision making:
-- Only lines with value "EQ" in the "series" column should be processed.
-- As the first stage, filter out all the lines that do not fulfil this criteria.
-- For every stock(with value "EQ" in the "series"), for every year, calculate the statistical parameters
-- (Minimum, Maximum, Mean and Standard Deviation) and store the generated information in properly
designated tables.

-- start of codes
-- one time jobs
-- **## Please avoid lines here on forward if nsestocksdb is available in 'show databases' command and
contains data                   ##**
-- ** and jump to the section of analysis **
create database nsestocksdb;
use nsestocksdb;
--create dataset
create table data_raw_headless
(SYMBOL string,SERIES string,OPEN float,HIGH float,LOW float,CLOSE float,LAST float,PREVCLOSE float,
TOTTRDQTY int,TOTTRDVAL float,TIMESTAMPS string,TOTALTRADES int,ISIN string)
row format delimited fields terminated by ',' lines terminated by '\n'
tblproperties("skip.header.line.count"="1");
load data inpath 'hdfs://localhost:9000/user/hive/warehouse/FINAL_FROM_DF.csv' into table
data_raw_headless;
-- find number of datapoints
select count(*) from data_raw_headless where isin!='';
-- result 846404

-- analysis jobs
--
create table pre_coll2 as select * from data_raw_headless where series=='EQ';
create table spc_coll2 (symbol string,timeval bigint,w_field float);
insert into spc_coll2 select symbol,unix_timestamp(timestamps,'yyyy-MM-dd'),close from pre_coll2;
create table finalcoll2 (symbol string,year int,w_field float);
insert into finalcoll2 select symbol,year(from_unixtime(timeval)) as year,w_field from spc_coll2;
create table anlysjob2 as select symbol,year,min(w_field),max(w_field),avg(w_field),stddev_pop(w_field)
from finalcoll2 group by symbol,year order by symbol;

-- obtained result
--hive> select * from  anlysjob2 limit 10;
--OK
--20MICRONS       2016    25.45   43.15   32.56518223411158       4.449799086240495
--20MICRONS       2017    33.7    62.7    41.634072642172534      6.590982144829505
--3IINFOTECH      2016    3.8     6.8     5.012348183736145       0.7338403196788177
--3IINFOTECH      2017    3.7     8.0     4.663104832172394       0.763375120711009
--3MINDIA 2016    9521.5  14939.55        12146.576930984313      1292.3010060900851
--3MINDIA 2017    10789.9 19366.4 13443.495959866432      1687.908570027977
--5PAISA  2017    187.3   388.75  283.72618902297245      67.21481238608698
--63MOONS 2017    54.9    159.65  84.53957428627825       23.64905316934828
--8KMILES 2016    591.3   2483.7  1646.2582968275556      541.7589068616428
--8KMILES 2017    369.5   987.9   613.1727826518397       138.32717290909883
--Time taken: 0.172 seconds, Fetched: 10 row(s)
--hive>

--****  result has 3345 elements result restricted to 10 elements
```

Pig Codes

```
/*Analysis 2:
 Calculation of various statistical quantities and decision making:
 Only lines with value "EQ" in the "series" column should be processed.
 As the first stage, filter out all the lines that do not fulfil this criteria.
 For every stock(with value "EQ" in the "series"), for every year,
 calculate the statistical parameters(Minimum, Maximum, Mean and Standard Deviation)
 and store the generated information in properly designated tables. */

-- start of code

--raw data load with defined schema
data_raw= LOAD '/home/kali/Hadoop/Local_Datasets/FINAL_FROM_DF.csv' USING PigStorage(',') as
(SYMBOL:chararray,SERIES:chararray,OPEN:float,HIGH:float,LOW:float,CLOSE:float,LAST:float,PREVCLOSE:float,TOTTRDQTY:int,TOTTRDVAL:float,TIMESTAMPS:Datetime,TOTALTRADES:int,ISIN:chararray);
data_collect = FILTER data_raw BY (OPEN>=0); --cleaning with condition (collect = headless)
rawGrp = GROUP data_raw ALL; --checker group
cltGrp = GROUP data_collect ALL; --checker group
ctrrw = FOREACH rawGrp GENERATE COUNT(data_raw);  -- counter value (846405)
ctrclt = FOREACH clnGrp GENERATE COUNT(data_collect); -- counter value (846404)
dump ctrrw --trigger calculation
dump ctrclt --trigger calculation

--no visible debris ... collect-> headless
--
--


--analysis job
pre_coll2 = FILTER data_collect BY (SERIES=='EQ');   -- prefilter with SERIES equals 'EQ'
finalcoll2 = FOREACH pre_coll2 GENERATE SYMBOL,GetYear(TIMESTAMPS) as (YEAR:int),CLOSE as
(W_field:float), CLOSE*CLOSE as (W_field2:float);  -- collection of required fields
calc_grp = GROUP finalcoll2 BY (SYMBOL,YEAR);  -- calculation group
-- finalizing required calculations
anlysjob2_ = FOREACH calc_grp GENERATE group as
gp,MIN(finalcoll2.W_field),MAX(finalcoll2.W_field),AVG(finalcoll2.W_field),SQRT(SUM(finalcoll2.W_field2)/C
OUNT(finalcoll2.W_field2) - AVG(finalcoll2.W_field)*AVG(finalcoll2.W_field));
anlysjob2 = ORDER anlysjob2_ BY SYMBOL;  -- final order

 --Storage
 STORE anlysjob2 INTO '/home/kali/Hadoop/Results/pig_results2/analysis2/' USING PigStorage();
 --


/*
Solution stored limited to 10 results
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results2/analysis2/part* |head -n 10
(20MICRONS,2016)        25.45   43.15   32.56518223411158       4.449799144115362
(20MICRONS,2017)        33.7    62.7    41.634072642172534      6.59098207014905
(3IINFOTECH,2016)       3.8     6.8     5.012348183736145       0.7338402417837622
(3IINFOTECH,2017)       3.7     8.0     4.663104832172394       0.7633750476806549
(3MINDIA,2016)  9521.5  14939.55        12146.576930984313      1292.3010897787285
(3MINDIA,2017)  10789.9 19366.4 13443.495959866432      1687.908552108419
(5PAISA,2017)   187.3   388.75  283.72618902297245      67.2148122582015
(63MOONS,2017)  54.9    159.65  84.53957428627825       23.649053295859137
(8KMILES,2016)  591.3   2483.7  1646.2582968275556      541.7588940911268
(8KMILES,2017)  369.5   987.9   613.1727826518397       138.32717721849752
kali@kali:~$
*/
```

Spark Codes

```
// Analysis 2
//   Calculation of various statistical quantities and decision making:
// Only lines with value "EQ" in the "series" column should be processed.
// As the first stage, filter out all the lines that do not fulfil this criteria.
// For every stock(with value "EQ" in the "series"), for every year, calculate the statistical parameters
// (Minimum, Maximum, Mean and Standard Deviation) and store the generated information in properly
designated tables.
```

```
//Start of Code
//cleaning of visible debris
//import
var data_raw= sc.textFile("hdfs://localhost:9000/assign2/FINAL_FROM_DF.csv")
var data_split=data_raw.map(x=>x.split(',')) //split by csv terms
var data_headless=data_split.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1) else iter }
//removing header line
data_headless.count // count confirm by 846404

//visible debris not noted ->  no additional filters imposed
//Data Scheme: SYMBOL    SERIES    OPEN     HIGH      LOW      CLOSE    LAST      PREVCLOSE
             TOTTRDQTY        TOTTRDVAL        TIMESTAMP        TOTALTRADES     ISIN

//operative methods
import Numeric.Implicits._
def mean[T: Numeric](xs: Iterable[T]): Double = xs.sum.toDouble / xs.size      //mean
def variance[T: Numeric](xs: Iterable[T]): Double = {
  val avg = mean(xs)                              //variance
  xs.map(_.toDouble).map(a => math.pow(a - avg, 2)).sum / xs.size
}
def stdDev[T: Numeric](xs: Iterable[T]): Double = math.sqrt(variance(xs))        //standard deviation

//analysis job
//
var precollect=data_headless.filter(x=>{x(1) == "EQ"})  //task1
var collectmap=precollect.map(x=>(x(0)+"\t- "+x(10).substring(0,4),x(5).toDouble))  //task2
var analysis2=collectmap.groupByKey.map{ case (k,v)=>(k,v.min,v.max,mean(v),stdDev(v))}.sortBy(_._1)
//task3 and final result
analysis2.saveAsTextFile("hdfs://localhost:9000/assign2/spark_jobs/analysis2")  //store

// end of code

//Result obtained :
//kali@kali:~$ hdfs dfs -cat /assign2/spark_jobs/analysis2/part* | head -n 10
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-26 19:28:57,232 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
//(20MICRONS     - 2016,25.45,43.15,32.56518218623481,4.449798961387773)
//(20MICRONS     - 2017,33.7,62.7,41.63407258064514,6.590982227296511)
//(3IINFOTECH    - 2016,3.8,6.8,5.012348178137656,0.7338403174843332)
//(3IINFOTECH    - 2017,3.7,8.0,4.663104838709679,0.7633751172614981)
//(3MINDIA       - 2016,9521.5,14939.55,12146.576923076926,1292.3010059868957)
//(3MINDIA       - 2017,10789.9,19366.4,13443.495967741936,1687.9086040389095)
//(5PAISA   - 2017,187.3,388.75,283.7261904761904,67.21481309780786)
//(63MOONS       - 2017,54.9,159.65,84.53957446808509,23.64905305624774)
//(8KMILES       - 2016,591.3,2483.7,1646.258299595143,541.7589089129798)
//(8KMILES       - 2017,369.5,987.9,613.172782258065,138.3271731430923)
```

```
//cat: Unable to write to output stream.
//cat: Unable to write to output stream.
//kali@kali:~$
//**
```

Question 3

Hive Codes

```
-- Analysis 3
--  Select any year for which data is available:
-- For the selected year, create a table that contains data only for those stocks that have an average total traded qntity of 3 lakhs or more per day.
-- Print out the first 25 entries of the table and submit.
-- From above output, select any 10 stocks from IT ('HCLTECH', 'NIITTECH', 'TATAELXSI','TCS', 'INFY', 'WIPRO', 'DATAMATICS','TECHM','MINDTREE' and 'OFSS')
-- and create a table combining their data. Find out the Pearsons Correlation Coeffecient for every pair of stocks you have selected above.
-- Final output should be in decreasing order of the coeffecient

-- start of codes
-- one time jobs
-- **## Please avoid lines here on forward if nsestocksdb is available in 'show databases' command and contains data                  ##**
-- ** and jump to the section of analysis **
create database nsestocksdb;
use nsestocksdb;
--create dataset
create table data_raw_headless
(SYMBOL string,SERIES string,OPEN float,HIGH float,LOW float,CLOSE float,LAST float,PREVCLOSE float, TOTTRDQTY int,TOTTRDVAL float,TIMESTAMPs string,TOTALTRADES int,ISIN string)
row format delimited fields terminated by ',' lines terminated by '\n'
tblproperties("skip.header.line.count"="1");
load data inpath 'hdfs://localhost:9000/user/hive/warehouse/FINAL_FROM_DF.csv' into table data_raw_headless;
-- find number of datapoints
select count(*) from data_raw_headless where isin!='';
-- result 846404

-- analysis jobs
--
create table preset31 as select * from data_raw_headless where year(from_unixtime(unix_timestamp(timestamps,'yyyy-MM-dd')))== 2017;
create table anlysjob3a as select * from preset31 where tottrdqty>=300000;
-- first 25 entries
select * from anlysjob3a limit 25;
--10 IT stocks
```

```
create table anlysjob3b as select * from anlysjob3a
where symbol =='HCLTECH' OR symbol == 'NIITTECH' OR symbol == 'TATAELXSI' OR symbol == 'TCS'
OR symbol == 'INFY' OR symbol == 'WIPRO'
OR symbol == 'DATAMATICS' OR symbol == 'TECHM' OR symbol == 'MINDTREE' OR symbol == 'OFSS';
create table it3left as select symbol,close from anlysjob3b;
create table it3right as select symbol,close from anlysjob3b;
create table crosscoll3 (sym1 string,sym2 string,val1 float,val2 float);
insert into crosscoll3 select it3left.symbol, it3right.symbol, it3left.close,it3right.close from it3left cross join
it3right
where it3left.symbol < it3right.symbol;
create table anlysjob3(sym1 string,sym2 string,rho double);
insert into anlysjob3 select sym1,sym2,(avg(val1*val2)-
(avg(val1)*avg(val2)))/(stddev_pop(val1)*stddev_pop(val2)) as rho from crosscoll3
group by sym1,sym2 order by rho desc;



-- obtained result
--
--hive> select * from anlysjob3 limit 10;
--OK
--HCLTECH  OFSS    0.060179274588645304
--HCLTECH  INFY    0.03636273681709574
--HCLTECH  MINDTREE    0.03605078081500838
--HCLTECH  TCS   0.03148815953621653
--MINDTREE  OFSS   0.027856217844901495
--HCLTECH  NIITTECH    0.020602642393445326
--HCLTECH  TECHM   0.017544870699555067
--MINDTREE  TCS    0.01698240987202918
--MINDTREE  NIITTECH   0.009471649548329374
--MINDTREE  TECHM    0.008152816287473285
--Time taken: 0.311 seconds, Fetched: 10 row(s)
--hive>
--

--*** output restricted to 10 results out of 45 rows.***
```

Pig Codes

```
/*Analysis 3:
Select any year for which data is available:
--> For the selected year, create a table that contains data only for those stocks that have an average total
traded quntity of 3 lakhs or more per day.
--> Print out the first 25 entries of the table and submit.
--> From above output, select any 10 stocks from IT ('HCLTECH', 'NIITTECH', 'TATAELXSI','TCS', 'INFY',
'WIPRO', 'DATAMATICS','TECHM','MINDTREE' and 'OFSS')
	and create a table combining their data. Find out the Pearsons Correlation Coeffecient for every pair
of stocks you have selected above.
```

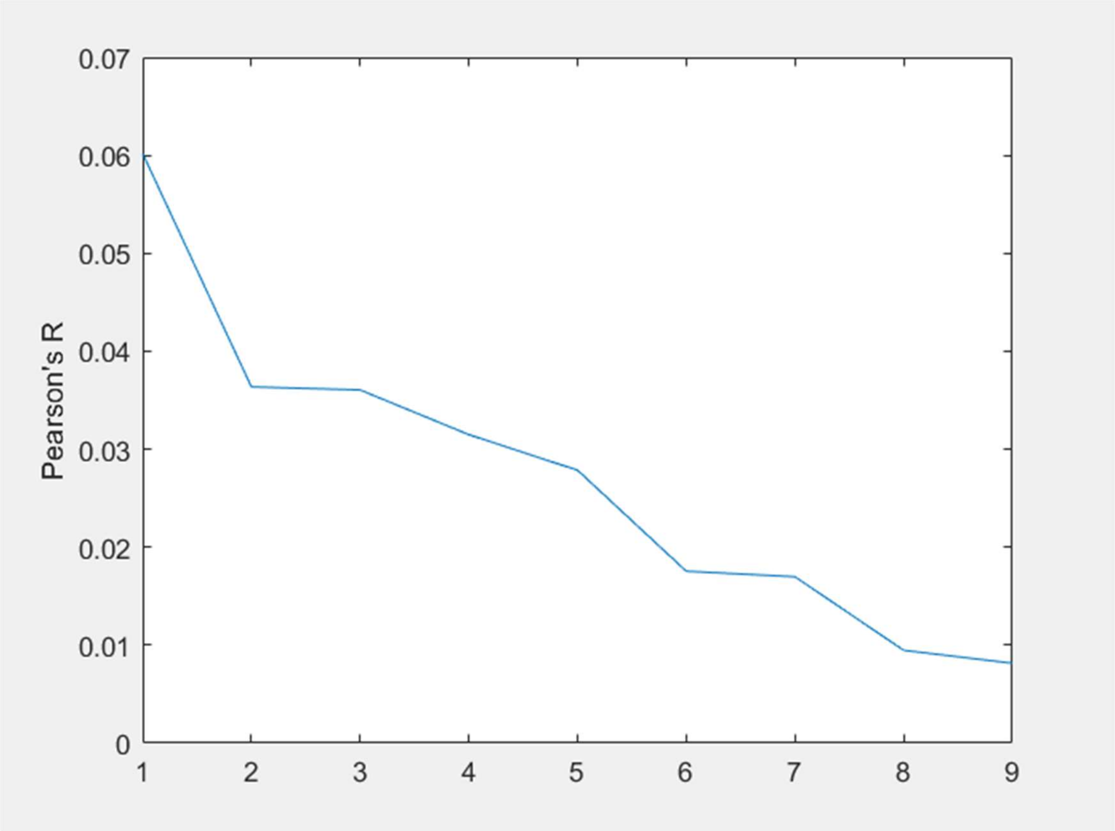
```
--> Final output should be in decreasing order of the coeffecient */

-- start of code
--raw data load with defined schema
data_raw= LOAD '/home/kali/Hadoop/Local_Datasets/FINAL_FROM_DF.csv' USING PigStorage(',') as
(SYMBOL:chararray,SERIES:chararray,OPEN:float,HIGH:float,LOW:float,CLOSE:float,LAST:float,PREVCLOSE:fl
oat,TOTTRDQTY:int,TOTTRDVAL:float,TIMESTAMPS:Datetime,TOTALTRADES:int,ISIN:chararray);
data_collect = FILTER data_raw BY (OPEN>=0); --cleaning with condition (collect = headless)
rawGrp = GROUP data_raw ALL; --checker group
cltGrp = GROUP data_collect ALL; --checker group
ctrrw = FOREACH rawGrp GENERATE COUNT(data_raw);  -- counter value (846405)
ctrclt = FOREACH cltGrp GENERATE COUNT(data_collect); -- counter value (846404)
dump ctrrw --trigger calculation
dump ctrclt --trigger calculation

--no visible debris ... collect-> headless
--
--


--analysis job
-- selected year 2017
precoll3 = FILTER data_collect BY GetYear(TIMESTAMPS)==2017;  -- filter by selected year
anlysjob3a = FILTER precoll3 BY TOTTRDQTY>=300000;   -- filter by specified criteria
f25_anlysjob3a = LIMIT anlysjob3a 25;  --  test variable to get first 25 entries in 'anlysjob3a'
STORE f25_anlysjob3a INTO '/home/kali/Hadoop/Results/pig_results2/analysis3/a/' USING PigStorage();  -
- store cycle 1
dump f25_anlysjob3a;  -- dumped to release values
--  ** filtration by 10 specified IT labels in SYMBOL column
anlysjob3b = FILTER anlysjob3a BY SYMBOL IN ('HCLTECH', 'NIITTECH', 'TATAELXSI','TCS', 'INFY', 'WIPRO',
'DATAMATICS','TECHM','MINDTREE','OFSS');
it3left = FOREACH anlysjob3b GENERATE SYMBOL,CLOSE; -- left joiner
it3right = FOREACH anlysjob3b GENERATE SYMBOL,CLOSE; -- right joiner
crosscoll3 = FILTER (CROSS it3left,it3right) BY  it3left.SYMBOL < it3right.SYMBOL;  --cross collection with
filtration by inequal symbol criteria
-- arrangement as per requirement
arranged = FOREACH crosscoll3 GENERATE CONCAT(it3left.SYMBOL ,'  ,  ',it3right.SYMBOL) as
(sym1n2:chararray),it3left.CLOSE as (val1:float),it3right.CLOSE as (val2:float),
it3left.CLOSE*it3left.CLOSE as (val1p2:float),it3right.CLOSE*it3right.CLOSE as
(val2p2:float),it3left.CLOSE*it3right.CLOSE as (val12:float);
arng_grp = GROUP arranged BY sym1n2;  -- grouping
-- final solution
anlysjob3_ = FOREACH arng_grp GENERATE group,(AVG(arranged.val12)-
(AVG(arranged.val1)*AVG(arranged.val2)))/(SQRT(SUM(arranged.val1p2)/COUNT(arranged.val1p2) -
AVG(arranged.val1)*AVG(arranged.val1))*SQRT(SUM(arranged.val2p2)/COUNT(arranged.val2p2) -
AVG(arranged.val2)*AVG(arranged.val2))) as (rho:float);
anlysjob3 = ORDER anlysjob3_ BY rho;
STORE anlysjob3 INTO '/home/kali/Hadoop/Results/pig_results2/analysis3/b/' USING PigStorage();  --
store cycle 2
```

```
/*
Results obtained
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results2/analysis3/b/part*
(HCLTECH , OFSS,0.060179274588645304)
(HCLTECH , INFY,0.03636273681709574)
(HCLTECH , MINDTREE,0.03605078081500838)
(HCLTECH , TCS,0.03148815953621653)
(MINDTREE , OFSS,0.027856217844901495)
(HCLTECH , NIITTECH,0.020602642393445326)
(HCLTECH , TECHM,0.017544870699555067)
(MINDTREE , TCS,0.01698240987202918)
(MINDTREE , NIITTECH,0.009471649548329374)
(MINDTREE , TECHM,0.008152816287473285)
kali@kali:~$
*/
```

Spark Codes

```
// Analysis 3
//  Select any year for which data is available:
// For the selected year, create a table that contains data only for those stocks that have an average total traded quntity of 3 lakhs or more per day.
// Print out the first 25 entries of the table and submit.
// From above output, select any 10 stocks from IT ('HCLTECH', 'NIITTECH', 'TATAELXSI','TCS', 'INFY', 'WIPRO', 'DATAMATICS','TECHM','MINDTREE' and 'OFSS')
// and create a table combining their data. Find out the Pearsons Correlation Coeffecient for every pair of stocks you have selected above.
//Final output should be in decreasing order of the coeffecient

//Start of Code
//cleaning of visible debris
//import
var data_raw= sc.textFile("hdfs://localhost:9000/assign2/FINAL_FROM_DF.csv")
var data_split=data_raw.map(x=>x.split(',')) //split by csv terms
var data_headless=data_split.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1) else iter }
//removing header line
data_headless.count // count confirm by 846404

//visible debris not noted ->  no additional filters imposed
//Data Scheme: SYMBOL    SERIES    OPEN    HIGH    LOW    CLOSE    LAST    PREVCLOSE
        TOTTRDQTY    TOTTRDVAL    TIMESTAMP    TOTALTRADES    ISIN

//operative methods
import Numeric.Implicits._
def mean[T: Numeric](xs: Iterable[T]): Double = xs.sum.toDouble / xs.size      //mean
def variance[T: Numeric](xs: Iterable[T]): Double = {
  val avg = mean(xs)                              //variance
  xs.map(_.toDouble).map(a => math.pow(a - avg, 2)).sum / xs.size
}
def stdDev[T: Numeric](xs: Iterable[T]): Double = math.sqrt(variance(xs))        //standard deviation

//analysis job
//
var preset31=data_headless.filter{x=> if(x(10).substring(0,4).toInt == 2017) true else false}   // prefilter by
year = 2017
var anlysjob3a=preset31.filter{x=> if(x(8).toFloat>=300000) true else false}    // filter by
tottdrval>=300000
var res1x=anlysjob3a.take(25)  //submit as in question
sc.parallelize(res1x).saveAsTextFile("hdfs://localhost:9000/assign2/spark_jobs/analysis3/Job1")
var anlysjob3b=anlysjob3a.filter{x=> if(x(0) =="HCLTECH" || x(0) == "NIITTECH" || x(0) == "TATAELXSI" ||
x(0) == "TCS" || x(0) == "INFY" || x(0) == "WIPRO" || x(0) == "DATAMATICS" || x(0) == "TECHM" || x(0) ==
"MINDTREE" || x(0) == "OFSS") true else false}
var it3left=anlysjob3b.map(x=>(x(0),x(5).toFloat))
var it3right=anlysjob3b.map(x=>(x(0),x(5).toFloat))
var cross=it3left.cartesian(it3right).map{case((a, b), (c, d))=>(a,c,b,d)}.filter{x=>(x._1 <
x._2)}.map{case((a,b,c,d))=>(a+" , "+b ,c,d,c*d)}
var gpsx_L=cross.map(x=>(x._1,x._2)).groupByKey  // iterable left  -> x
var gpsx_R=cross.map(x=>(x._1,x._3)).groupByKey  // iterable right -> y
var gpsx_C=cross.map(x=>(x._1,x._4)).groupByKey  // iterable center -> (x*y)
var joinback=gpsx_L.join(gpsx_R).join(gpsx_C)  // join back grouped iterables
var analysis3b_f=joinback.map{case(w, ((x, y), z))=> (w,(mean(z)-
mean(x)*mean(y))/(stdDev(x)*stdDev(y)))}.sortBy(_._2,false)  // final operation
analysis3b_f.saveAsTextFile("hdfs://localhost:9000/assign2/spark_jobs/analysis3/Job2")  //store

// end of code

//Result obtained :
//kali@kali:~$ hdfs dfs -cat /assign2/spark_jobs/analysis3/Job2/part* | head -n 10
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-27 01:29:20,040 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
//(HCLTECH , OFSS,0.060179274588645304)
//(HCLTECH , INFY,0.03636273681709574)
//(HCLTECH , MINDTREE,0.03605078081500838)
//(HCLTECH , TCS,0.03148815953621653)
//(MINDTREE , OFSS,0.027856217844901495)
//(HCLTECH , NIITTECH,0.020602642393445326)
//(HCLTECH , TECHM,0.017544870699555067)
//(MINDTREE , TCS,0.01698240987202918)
//(MINDTREE , NIITTECH,0.009471649548329374)
//(MINDTREE , TECHM,0.008152816287473285)
//cat: Unable to write to output stream.
//cat: Unable to write to output stream.
//cat: Unable to write to output stream.
```

```
//kali@kali:~$
//
```



Question 4

Python solution

```
#!usr/bin/python
# Analysis4:
#  Use the coorrelation information generated in step 3 in the following way:
# a. Assume you have Rs10 lakh to invest
# b. Assume you have to invest in six stocks on the first working day of January of the next year.
# c. By using logic/simulation/etc. Identify the stocks that you will invest in,
#       such that at the end of the year: At least your overall capital (Rs 10 lakh) is protected.

import pandas as pd

df = pd.read_csv(r"/home/kali/Hadoop/Local_Datasets/FINAL_FROM_DF.csv",sep='\t')
df.columns = ['SYMBOL','MIN','MAX','AVG','STDEV','YEAR']

dfcor = pd.read_csv(r"/home/kali/Hadoop/Local_Datasets/FINAL_FROM_DF.csv",sep='\t',header=None)

l = []
```

```
lrate = []
for i in dfcor.iloc[:,0]:
    avg2011 = int(df[(df['SYMBOL'] == i) & (df['YEAR'] == 2011)]['AVG'])
    avg2013 = int(df[(df['SYMBOL'] == i) & (df['YEAR'] == 2013)]['AVG'])
    l.append(((avg2013-avg2011)/avg2011)*100)
    lrate.append()

dfcor.insert(1,'5',l)

dfcor.columns = ['SYMBOL1','%GROWTH',"SYMBOL2",'CORR_BW_S1andS2']

dfcor[dfcor['%GROWTH'] > 50].sort_values('%GROWTH')['SYMBOL1'].unique()
#Out[91]: array(['TCS', 'TECHM', 'HCLTECH', 'MINDTREE'], dtype=object)
#This list is in ascending order of GROWTH

dfcor[dfcor['%GROWTH'] > 50]
dfnse =
pd.read_csv(r"/home/kali/Hadoop/Local_Datasets/FINAL_FROM_DF.csv",usecols=[0,1,2,3,4,5,6,7,8,9,10])

#LIST OF STOCKS THAT I WILL BUY
lbuy = ['MINDTREE','TCS','INFY','OFSS','TECHM' , 'HCLTECH']

lbuy2014JAN = []
lbuy2014DEC = []

for i in lbuy:
    jan2014 = int(dfnse[(dfnse['SYMBOL']==i) & (dfnse['TIMESTAMP'] == '01-JAN-2014')]['CLOSE'])
    dec2014 = int(dfnse[(dfnse['SYMBOL']==i) & (dfnse['TIMESTAMP'] == '01-DEC-2014')]['CLOSE'])
    lbuy2014JAN.append(jan2014)
    lbuy2014DEC.append(dec2014)

lbuy
#Out[115]: ['MINDTREE', 'TCS', 'INFY', 'OFSS', 'TECHM', 'HCLTECH']


lbuy2014JAN
#Out[116]: [1549, 2153, 3468, 3274, 1828, 1258]

lbuy2014DEC
#Out[117]: [1244, 2692, 4349, 3444, 2653, 1671]
```

Conclusion

This project is done in the manner of an analysis and no conclusion could be drawn to a certain limit as we are not deducing any solution from this project, but in-fact pulling up statistics which are impossible to handle by conventional methods. All deducible solutions are provided with the curves where possible. The non-graphed results are just provided for tallying reasons. Question 4 specifically is troubling to obtain by Hive, Pig or Spark and the modularity of Python MapReduce was more suitable hence that method is followed

All codes pertaining to this project is available on https://github.com/WolfDev8675/RepoSJX7/tree/Assign2

Bibliography

https://hadoop.apache.org/

https://pig.apache.org/

https://hive.apache.org/

https://spark.apache.org/

https://gethue.com/

https://www.mongodb.com/

https://mariadb.org/

https://www.packtpub.com/product/learning-hadoop-2/9781783285518

https://www.scala-lang.org/