****

**Developed By: Manav Patadia**

**Registration Num.: PGDDSPJULY2020/04**

**Email ID: manavpatadia@gmail.com**

Case Study 2

## Stock Market

## Acknowledgement

I would like to thank BSE Institute for providing me with training in Data Science and an opportunity to deepen my knowledge of Big Data Tools.

I would also like to thank Ashok Gupta, professor for big data and analytics solutions at BSE Institute, for providing me support in big data analytics project and training me in Big data tools like Hive, Map Reduce, Hbase, Pig and Spark.

## Table of Contents

[Retail Sales Error! Bookmark not defined.](file:///G:\PGD_class\Retail-Sales-Big-Data\Retail%20Sales%20Big%20Data%20Project%20-%20Copy.docx#_Toc56559452)

[Acknowledgement 2](#_Toc56559453)

[Table of Contents 3](#_Toc56559454)

[Objective & Purpose 4](#_Toc56559455)

[Functionality 5](#_Toc56559456)

[Hardware & Software Requirements 6](#_Toc56559457)

[Why Hadoop 7](#_Toc56559458)

[Big Data Tools 8](#_Toc56559459)

[Future Scope of Big Data 9](#_Toc56559460)

[Data Preparation 10](#_Toc56559461)

[Question 1 11](#_Toc56559462)

[Solution 1: 11](#_Toc56559463)

[Question 2 15](#_Toc56559464)

[Solution 2: 15](#_Toc56559465)

[Question 3 19](#_Toc56559466)

[Solution 3: 20](#_Toc56559467)

[Question 4 Error! Bookmark not defined.](#_Toc56559468)

[Solution 4: Error! Bookmark not defined.](#_Toc56559469)

[Question 5 Error! Bookmark not defined.](#_Toc56559470)

[Solution 5: Error! Bookmark not defined.](#_Toc56559471)

[Question 6 Error! Bookmark not defined.](#_Toc56559472)

[Solution 6: Error! Bookmark not defined.](#_Toc56559473)

[Question 7 Error! Bookmark not defined.](#_Toc56559474)

[Solution 7: Error! Bookmark not defined.](#_Toc56559475)

## Objective & Purpose

**Below are the objectives and purpose of this Case Study:**

* Analyse Retail Sales Data Set
* Clean Data: Remove bad records
* Prepare solution for each 7 questions in Hive, Pig, Map-Reduce and Spark.
* Store output in either local file system or in HDFS.

## Functionality

1. **Use the given csv file as input data and implement following transformations:**

* **Filter Rows on specified criteria "Symbol equals GEOMETRIC":** Display records where symbol is Geometric.
* **Select specific columns from those available: SYMBOL, OPEN, HIGH, LOW and CLOSE which meets above criteria:** Display SYMBOL, OPEN, HIGH, LOW and CLOSE columns where symbol is Geometric.
* **Generate count of the number of rows from above result:** Display number of records generated from above query.

1. **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:** Display records where series is EQ.
* **For every stock, for every year, calculate the following statistical parameters: Minimum, Maximum, Mean and Standard Deviation and store the generated information in properly designated tables:** From above records, calculate minimum, maximum, mean and standard deviation and store it in table.

1. **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 total traded quntity of 3 lakhs or more per day. Print out the first 25 entries of the table and submit:** Display 25 records where TOTTRDQTY >= 300000 and records are from a single selected year.
* **From among these, select any 10 stocks from IT ('HCLTECH', 'NIITTECH', 'TATAELXSI','TCS', 'INFY', 'WIPRO', 'DATAMATICS','TECHM','MINDTREE' and 'OFSS') and create a table combining their data:** From above records, select rows where SYMBOL is: HCLTECH or NIITTECH or TATAELXSI or TCS or INFY or WIPRO or DATAMATICS or TECHM or MINDTREE or OFSS and store it in a table.
* **Find out the Pearsons Correlation Coeffecient for every pair of stocks you have selected. Final output should be in decreasing order of the coefficient:** From above records, calculate Pearsons Correlation Coeffecient for every pair of symbol.

## Hardware & Software Requirements

**Below are the minimum hardware requirements of this project:**

* i5 Processor
* 8 GB RAM
* 50 GB Hard Drive Space

**Below are the software requirements of this project:**

* Oracle Virtual Box and Ubuntu (64-bit) configured in it
* Download and configure JDK, Hadoop, Hive, Hbase, Pig and Spark

## Why Hadoop

**1. Scalable**

Hadoop is a highly scalable storage platform, because it can store and distribute very large data sets across hundreds of inexpensive servers that operate in parallel. Unlike traditional relational database systems (RDBMS) that can't scale to process large amounts of data, Hadoop enables businesses to run applications on thousands of nodes involving thousands of terabytes of data.

**2. Cost effective**

Hadoop also offers a cost effective storage solution for businesses' exploding data sets. The problem with traditional relational database management systems is that it is extremely cost prohibitive to scale to such a degree in order to process such massive volumes of data. The raw data would be deleted, as it would be too cost-prohibitive to keep. While this approach may have worked in the short term, this meant that when business priorities changed, the complete raw data set was not available, as it was too expensive to store. Hadoop, on the other hand, is designed as a scale-out architecture that can affordably store all of a company's data for later use. The cost savings are staggering: instead of costing thousands to tens of thousands of pounds per terabyte, Hadoop offers computing and storage capabilities for hundreds of pounds per terabyte.

**3. Flexible**

Hadoop enables businesses to easily access new data sources and tap into different types of data (both structured and unstructured) to generate value from that data. This means businesses can use Hadoop to derive valuable business insights from data sources such as social media, email conversations or clickstream data. In addition, Hadoop can be used for a wide variety of purposes, such as log processing, recommendation systems, data warehousing, market campaign analysis and fraud detection.

**4. Fast**

Hadoop's unique storage method is based on a distributed file system that basically 'maps' data wherever it is located on a cluster. The tools for data processing are often on the same servers where the data is located, resulting in much faster data processing. If you're dealing with large volumes of unstructured data, Hadoop is able to efficiently process terabytes of data in just minutes, and petabytes in hours.

**5. Resilient to failure**

A key advantage of using Hadoop is its fault tolerance. When data is sent to an individual node, that data is also replicated to other nodes in the cluster, which means that in the event of failure, there is another copy available for use.

The MapR distribution goes beyond that by eliminating the NameNode and replacing it with a distributed No NameNode architecture that provides true high availability. Our architecture provides protection from both single and multiple failures.

When it comes to handling large data sets in a safe and cost-effective manner, Hadoop has the advantage over relational database management systems, and its value for any size business will continue to increase as unstructured data continues to grow.

## Big Data Tools

**Pig vs Hive**

* Hive Hadoop Component is used mainly by data analysts whereas Pig Hadoop Component is generally used by Researchers and Programmers.
* Hive Hadoop Component is used for completely structured Data whereas Pig Hadoop Component is used for structured and semi structured data.
* Hive Hadoop Component has a declarative SQLish language (HiveQL) which is easy to learn and requires less lines of code whereas Pig Hadoop Component has a procedural data flow language which is built for people who aren’t familliar with Java, Python or SQL and it requires more lines of code in compared to Hive.
* Pig supports Avro whereas Hive does not.
* Apache Pig is 36% faster than Apache Hive for join operations on datasets.
* Apache Pig is 46% faster than Apache Hive for arithmetic operations.
* Apache Pig is 10% faster than Apache Hive for filtering 10% of the data.
* Apache Pig is 18% faster than Apache Hive for filtering 90% of the data.

**Map-Reduce vs Spark**

* Linear processing of huge data sets. Hadoop MapReduce allows parallel processing of huge amounts of data. It breaks a large chunk into smaller ones to be processed separately on different data nodes and automatically gathers the results across the multiple nodes to return a single result. In casethe resulting dataset is larger than available RAM, Hadoop MapReduce may outperform Spark.
* Fast data processing. In-memory processing makes Spark faster than Hadoop MapReduce – up to 100 times for data in RAM and up to 10 times for data in storage.
* Iterative processing.If the task is to process data again and again – Spark defeats Hadoop MapReduce. Spark’s Resilient Distributed Datasets (RDDs) enablemultiple map operations in memory, while Hadoop MapReduce has to write interim results to a disk.
* Near real-time processing.If a business needs immediate insights, then they should opt for Spark and its in-memory processing.
* Graph processing. Spark’s computational model is good for iterative computations that are typical in graph processing. And Apache Spark has GraphX – an API for graph computation.
* Machine learning. Spark has MLlib – a built-in machine learning library, while Hadoop needs a third-party to provide it. MLlib has out-of-the-box algorithms that also run in memory.
* Joining datasets. Due to its speed, Spark can create all combinations faster, though Hadoop may be better if joining of very large data sets that requires alot of shuffling and sorting is needed.

## Future Scope of Big Data

* Visual data discovery tools will be growing 2.5 times faster than rest of the Business Intelligence (BI) market. By 2025, investing in this enabler of end-user self-service will become a requirement for all enterprises.
* Over the next five years spending on cloud-based Big Data and analytics (BDA) solutions will grow three times faster than spending for on-premise solutions. Hybrid on/off premise deployments will become a requirement.
* Shortage of skilled staff will persist. In the U.S. alone there will be 181,000 deep analytics roles in 2021 and five times that many positions requiring related skills in data management and interpretation.
* By 2017 unified data platform architecture will become the foundation of BDA strategy. The unification will occur across information management, analysis, and search technology.
* Growth in applications incorporating advanced and predictive analytics, including machine learning, will accelerate in 2021. These apps will grow 65% faster than apps without predictive functionality.
* 70% of large organizations already purchase external data and 100% will do so by 2022. In parallel more organizations will begin to monetize their data by selling them or providing value-added content.
* Adoption of technology to continuously analyse streams of events will accelerate in 2021 as it is applied to Internet of Things (IoT) analytics, which is expected to grow at a five-year compound annual growth rate (CAGR) of 30%.
* Decision management platforms will expand at a CAGR of 60% through 2021 in response to the need for greater consistency in decision making and decision making process knowledge retention.
* Rich media (video, audio, image) analytics will at least triple in 2021 and emerge as the key driver for BDA technology investment.
* By 2022 half of all consumers will interact with services based on cognitive computing on a regular basis.

## Data Preparation

### Sample Data

**HDFS Commands for Input and Output folders:**

hdfs dfs -mkdir /STOCK\_MARKET\_ASSIGNMENT

hdfs dfs -mkdir /STOCK\_MARKET\_ASSIGNMENT/INPUT

hdfs dfs -mkdir /STOCK\_MARKET\_ASSIGNMENT/OUTPUT

hdfs dfs -mkdir /STOCK\_MARKET\_ASSIGNMENT/OUTPUT/HIVE/

hdfs dfs -mkdir /STOCK\_MARKET\_ASSIGNMENT/OUTPUT/PIG/

hdfs dfs -mkdir /STOCK\_MARKET\_ASSIGNMENT/OUTPUT/SPARK/

hdfs dfs -put /home/manav/Documents/STOCK\_MARKET\_ASSIGNMENT/DATA/STOCK\_MARKET\_DS.csv /STOCK\_MARKET\_ASSIGNMENT/INPUT/

**HIVE Commands for Table Creation:**

CREATE DATABASE UC2\_STOCK;

USE UC2\_STOCK;

CREATE EXTERNAL TABLE UC2\_STOCK.NSEDATA (

SYMBOL STRING, SERIES STRING, OPEN DOUBLE, HIGH DOUBLE, LOW DOUBLE, CLOSE DOUBLE, LAST DOUBLE, PREVCLOSE DOUBLE,

TOTTRDQTY INT, TOTTRVAL DOUBLE, MYDATE STRING, TOTALTRADES INT, ISIN STRING)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION '/STOCK\_MARKET\_ASSIGNMENT/INPUT';

## Question 1

**Use the given csv file as input data and implement following transformations:**

* **Filter Rows on specified criteria "Symbol equals GEOMETRIC":** Display records where symbol is Geometric.
* **Select specific columns from those available: SYMBOL, OPEN, HIGH, LOW and CLOSE which meets above criteria:** Display SYMBOL, OPEN, HIGH, LOW and CLOSE columns where symbol is Geometric.
* **Generate count of the number of rows from above result:** Display number of records generated from above query.

## Solution 1:

### Hive

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/HIVE/1\_1'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

SELECT \* FROM UC2\_STOCK.NSEDATA WHERE SYMBOL = "GEOMETRIC";

INSERT OVERWRITE DIRECTORY '/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/HIVE/1\_2'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

SELECT SYMBOL, OPEN, HIGH, LOW, CLOSE

FROM UC2\_STOCK.NSEDATA

WHERE SYMBOL = "GEOMETRIC";

INSERT OVERWRITE DIRECTORY '/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/HIVE/1\_3'

ROW FORMAT DELIMITED

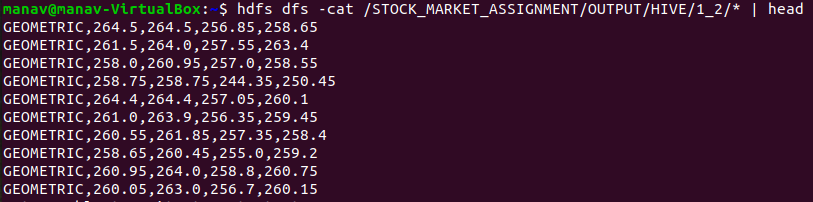
FIELDS TERMINATED BY ','

STORED AS TEXTFILE

SELECT COUNT(\*) AS NUM\_ROWS FROM UC2\_STOCK.NSEDATA WHERE SYMBOL = "GEOMETRIC";

##### G:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\HIVE_1_1.PNGOutput:

G:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\HIVE_1_3.PNG



### Pig

##### Pig Commands:

STOCK = LOAD 'hdfs://localhost:9000/STOCK\_MARKET\_ASSIGNMENT/INPUT/'

USING PigStorage(',') AS

( SYMBOL:CHARARRAY, SERIES:CHARARRAY, OPEN:DOUBLE, HIGH:DOUBLE, LOW:DOUBLE, CLOSE:DOUBLE, LAST:DOUBLE, PREVCLOSE:DOUBLE, TOTTRDQTY:DOUBLE, TOTTRDVAL:DOUBLE, TIMESTAMP:CHARARRAY, TOTALTRADES:INT, ISIN:CHARARRAY);

FILTER\_DATA = FILTER STOCK BY SYMBOL == 'GEOMETRIC';

STORE FILTER\_DATA INTO 'hdfs://localhost:9000/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/PIG/1\_1/' USING PigStorage (',');

IMP\_COL = FOREACH FILTER\_DATA GENERATE SYMBOL, OPEN, HIGH, LOW, CLOSE;

STORE IMP\_COL INTO 'hdfs://localhost:9000/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/PIG/1\_2/'

USING PigStorage (',');

GROUP\_ALL = GROUP IMP\_COL ALL;

COUNT\_FILTERED= FOREACH GROUP\_ALL GENERATE COUNT(IMP\_COL.SYMBOL) AS (COUNT\_GEO:LONG);

STORE COUNT\_FILTERED INTO 'hdfs://localhost:9000/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/PIG/1\_3/'

USING PigStorage (',');

##### G:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\PIG_1_2.PNGG:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\PIG_1_1.PNGOutput:

G:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\PIG_1_3.PNG

### Spark

val InputFile = "hdfs://localhost:9000/STOCK\_MARKET\_ASSIGNMENT/INPUT/STOCK\_MARKET\_DS.csv"

val OutputFile = "hdfs://localhost:9000/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/SPARK/"

val SYMBOL = 0

val SERIES = 1

val OPEN = 2

val HIGH = 3

val LOW = 4

val CLOSE = 5

val LAST = 6

val PREVCLOSE = 7

val TOTTRDQTY = 8

val TOTTRDVAL = 9

val TIMESTAMP = 10

val TOTALTRADES = 11

##### Spark Commands:

val STOCK = sc.textFile(InputFile).map(row => row.split(","))

val FilterMap = STOCK.filter(col => col(SYMBOL).toString == "GEOMETRIC").map(col => {(col(SYMBOL).toString, col(SERIES).toString, col(OPEN).toDouble, col(HIGH).toDouble, col(LOW).toDouble, col(CLOSE).toDouble, col(LAST).toDouble, col(PREVCLOSE).toDouble, col(TOTTRDQTY).toDouble, col(TOTTRDVAL).toDouble, col(TIMESTAMP).toString, col(TOTALTRADES).toLong, col(ISIN).toString)})

FilterMap.map(r => r.productIterator.mkString(",")).saveAsTextFile(OutputFile+"1\_1")

val STOCK = sc.textFile(InputFile).map(row => row.split(","))

val colMap = STOCK.filter(col => col(SYMBOL).toString == "GEOMETRIC").map(col => {(col(SYMBOL).toString, col(OPEN).toDouble, col(HIGH).toDouble, col(LOW).toDouble, col(CLOSE).toDouble)})

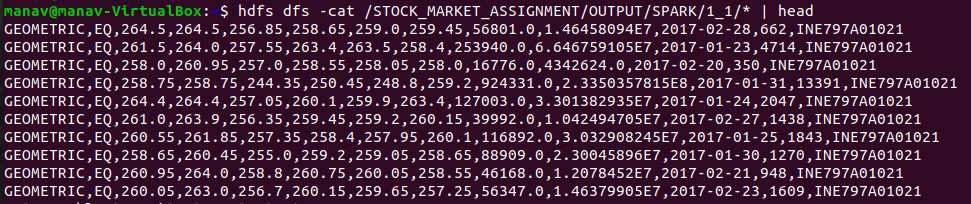
colMap.map(r => r.productIterator.mkString(",")).saveAsTextFile(OutputFile+"1\_2")

val STOCK = sc.textFile(InputFile).map(row => row.split(","))

val countRDD = sc.parallelize(Seq(colMap.count(),""))

countRDD.saveAsTextFile(OutputFile+"1\_3\_1")

##### G:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\SPARK_1_2.PNGOutput:

G:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\SPARK_1_3.PNG

## Question 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:** Display records where series is EQ.
* **For every stock, for every year, calculate the following statistical parameters: Minimum, Maximum, Mean and Standard Deviation and store the generated information in properly designated tables:** From above records, calculate minimum, maximum, mean and standard deviation and store it in table.

## Solution 2:

### Hive

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/HIVE/2\_1'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

SELECT \* FROM UC2\_STOCK.NSEDATA WHERE SERIES = 'EQ';

INSERT OVERWRITE DIRECTORY '/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/HIVE/2\_2'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

SELECT SYMBOL, MIN(CLOSE), MAX(CLOSE), ROUND(AVG(CLOSE),6), ROUND(STDDEV\_POP(CLOSE),6), SUBSTR(MYDATE,1,4) AS YEAR

FROM UC2\_STOCK.NSEDATA

WHERE SERIES = 'EQ'

GROUP BY SYMBOL, SUBSTR(MYDATE,1,4)

ORDER BY SYMBOL, YEAR DESC;

CREATE EXTERNAL TABLE UC2\_STOCK.STOCKSTATS

(SYMBOL STRING, MIN FLOAT, MAX FLOAT, MEAN FLOAT, STD FLOAT, YEAR STRING)

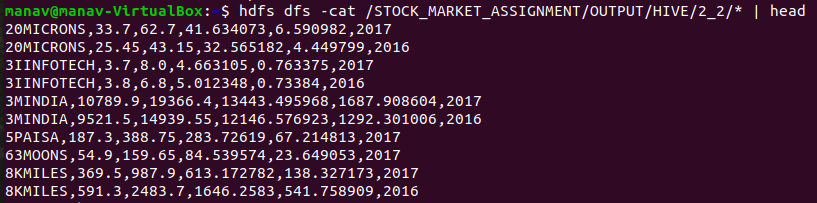
ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

LOCATION '/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/HIVE/2\_2';

##### G:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\HIVE_2_1.PNGOutput:



### Pig

STOCK = LOAD 'hdfs://localhost:9000/STOCK\_MARKET\_ASSIGNMENT/INPUT/'

USING PigStorage(',') AS

( SYMBOL:CHARARRAY, SERIES:CHARARRAY, OPEN:DOUBLE, HIGH:DOUBLE, LOW:DOUBLE, CLOSE:DOUBLE, LAST:DOUBLE, PREVCLOSE:DOUBLE, TOTTRDQTY:DOUBLE, TOTTRDVAL:DOUBLE, TIMESTAMP:CHARARRAY, TOTALTRADES:INT, ISIN:CHARARRAY);

##### Pig Commands:

FILTER\_DATA = FILTER STOCK BY SERIES == 'EQ';

STORE FILTER\_DATA INTO 'hdfs://localhost:9000/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/PIG/2\_1/'

USING PigStorage (',');

GROUP\_SYMBOL\_YEAR = GROUP FILTER\_DATA BY (SYMBOL, SUBSTRING(TIMESTAMP,0,4));

STATS\_ALL = FOREACH GROUP\_SYMBOL\_YEAR {

MINIMUM = MIN(FILTER\_DATA.CLOSE);

MAXIMUM = MAX(FILTER\_DATA.CLOSE);

MEAN = AVG(FILTER\_DATA.CLOSE);

CNT = COUNT(FILTER\_DATA.CLOSE);

CSQ = FOREACH FILTER\_DATA GENERATE CLOSE \* CLOSE AS (CC:DOUBLE);

GENERATE group.$0 AS (SYMBOL:CHARARRAY), MINIMUM AS (MIN:DOUBLE), MAXIMUM AS (MAX:DOUBLE), ROUND\_TO(MEAN,6) AS (MEAN:DOUBLE), ROUND\_TO(SQRT(SUM(CSQ.CC) / (CNT \* 1.0) - (MEAN \* MEAN)),6) AS (STDDEV:DOUBLE), group.$1 AS (YEAR:INT);

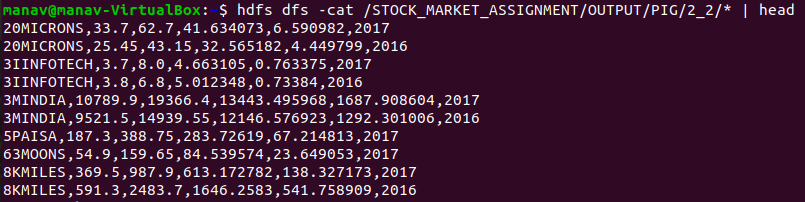
};

SORT\_STATS = ORDER STATS\_ALL BY SYMBOL, YEAR DESC;

STORE SORT\_STATS INTO 'hdfs://localhost:9000/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/PIG/2\_2/' USING PigStorage (',');

\

##### G:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\PIG_2_1.PNGOutput:



### Spark

##### Spark Commands:

val STOCK = sc.textFile(InputFile).map(row => row.split(","))

val FilterMap = STOCK.filter(col => col(SERIES).toString == "EQ").map(col => {(col(SYMBOL).toString, col(SERIES).toString, col(OPEN).toDouble, col(HIGH).toDouble, col(LOW).toDouble, col(CLOSE).toDouble, col(LAST).toDouble, col(PREVCLOSE).toDouble, col(TOTTRDQTY).toDouble, col(TOTTRDVAL).toDouble, col(TIMESTAMP).toString, col(TOTALTRADES).toLong, col(ISIN).toString)})

FilterMap.map(r => r.productIterator.mkString(",")).saveAsTextFile(OutputFile+"2\_1")

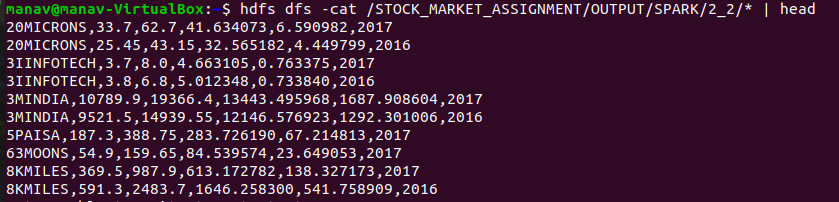
val stock\_map = FilterMap.map(col => {(col.\_1.toString +"\_"+ col.\_11.toString.split("-")(0),col.\_6)})

val stats\_all = stock\_map.groupByKey().mapValues(sq => (sq.min, sq.max, sq.sum/sq.size, org.apache.spark.util.StatCounter(sq).stdev))

val stats\_map = stats\_all.map(col => {(col.\_1.split("\_")(0), col.\_2.\_1, col.\_2.\_2, BigDecimal(col.\_2.\_3).setScale(6,BigDecimal.RoundingMode.HALF\_UP), BigDecimal(col.\_2.\_4).setScale(6, BigDecimal.RoundingMode.HALF\_UP), col.\_1.split("\_")(1))}).sortBy(\_.\_6,false).sortBy(\_.\_1)

stats\_map.map(r => r.productIterator.mkString(",")).saveAsTextFile(OutputFile+"2\_2")

##### G:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\SPARK_2_1.PNGOutput:



## Question 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 total traded quntity of 3 lakhs or more per day. Print out the first 25 entries of the table and submit:** Display 25 records where TOTTRDQTY >= 300000 and records are from a single selected year.
* **From among these, select any 10 stocks from IT ('HCLTECH', 'NIITTECH', 'TATAELXSI','TCS', 'INFY', 'WIPRO', 'DATAMATICS','TECHM','MINDTREE' and 'OFSS') and create a table combining their data:** From above records, select rows where SYMBOL is: HCLTECH or NIITTECH or TATAELXSI or TCS or INFY or WIPRO or DATAMATICS or TECHM or MINDTREE or OFSS and store it in a table.
* **Find out the Pearsons Correlation Coeffecient for every pair of stocks you have selected. Final output should be in decreasing order of the coefficient:** From above records, calculate Pearsons Correlation Coeffecient for every pair of symbol.

## Solution 3:

### Hive

#### Monthly

##### Hive Commands:

CREATE TABLE UC2\_STOCK.STOCK\_2017(

SYMBOL STRING, SERIES STRING, OPEN DOUBLE, HIGH DOUBLE, LOW DOUBLE,

CLOSE DOUBLE, LAST DOUBLE, PREVCLOSE DOUBLE, TOTTRDQTY INT, TOTTRVAL DOUBLE,

MYDATE STRING, TOTALTRADES INT, ISIN STRING);

INSERT OVERWRITE TABLE UC2\_STOCK.STOCK\_2017

SELECT \* FROM UC2\_STOCK.NSEDATA WHERE TOTTRDQTY >= 300000 AND SUBSTR(MYDATE,1,4) = '2017';

INSERT OVERWRITE DIRECTORY '/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/HIVE/3\_1'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

SELECT \* FROM UC2\_STOCK.STOCK\_2017 LIMIT 25;

CREATE TABLE UC2\_STOCK.ITSTOCK(

SYMBOL STRING, SERIES STRING, OPEN DOUBLE, HIGH DOUBLE, LOW DOUBLE,

CLOSE DOUBLE, LAST DOUBLE, PREVCLOSE DOUBLE, TOTTRDQTY INT, TOTTRVAL DOUBLE,

MYDATE STRING, TOTALTRADES INT, ISIN STRING);

INSERT OVERWRITE TABLE UC2\_STOCK.ITSTOCK

SELECT \* FROM UC2\_STOCK.STOCK\_2017

WHERE SYMBOL IN ('HCLTECH', 'NIITTECH', 'TATAELXSI','TCS', 'INFY', 'WIPRO', 'DATAMATICS','TECHM','MINDTREE', 'OFSS');

INSERT OVERWRITE DIRECTORY '/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/HIVE/3\_2'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE

SELECT \* FROM UC2\_STOCK.ITSTOCK;

CREATE TABLE UC2\_STOCK.PEARSONCORITSTOCK(SYMBOL1 STRING, SYMBOL2 STRING, CORR FLOAT);

INSERT OVERWRITE TABLE UC2\_STOCK.PEARSONCORITSTOCK

SELECT SYMBOL1, SYMBOL2,

ROUND((PSUM - (SUM1 \* SUM2 / N)) / SQRT((SUM1SQ - POW(SUM1, 2.0) / N) \* (SUM2SQ - POW(SUM2, 2.0) / N)),8) AS PEARSONCOEFFICIENT

FROM

(SELECT T1.SYMBOL AS SYMBOL1, T2.SYMBOL AS SYMBOL2,

SUM(T1.CLOSE) AS SUM1, SUM(T2.CLOSE) AS SUM2, SUM(T1.CLOSE \* T1.CLOSE) AS SUM1SQ, SUM(T2.CLOSE \* T2.CLOSE) AS SUM2SQ, SUM(T1.CLOSE \* T2.CLOSE) AS PSUM, COUNT(\*) AS N

FROM UC2\_STOCK.ITSTOCK T1

CROSS JOIN UC2\_STOCK.ITSTOCK T2

ON T1.MYDATE=T2.MYDATE

WHERE T1.SYMBOL > T2.SYMBOL

GROUP BY T1.SYMBOL, T2.SYMBOL

ORDER BY T1.SYMBOL ASC,T2. SYMBOL ASC, FROM\_UNIXTIME(UNIX\_TIMESTAMP(T1.MYDATE, 'yyyy-mm-dd'), 'yyyy-MMM-dd') ASC) STEP1

ORDER BY PEARSONCOEFFICIENT DESC;

INSERT OVERWRITE DIRECTORY '/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/HIVE/3\_3'

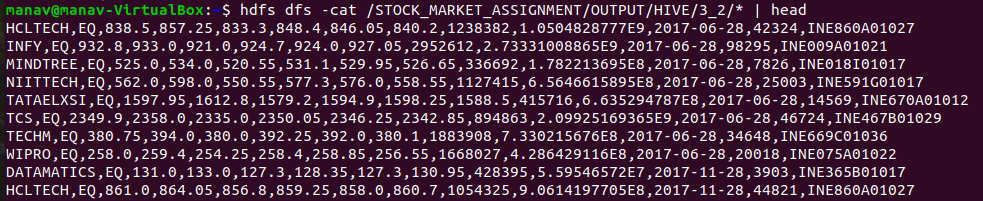
ROW FORMAT DELIMITED

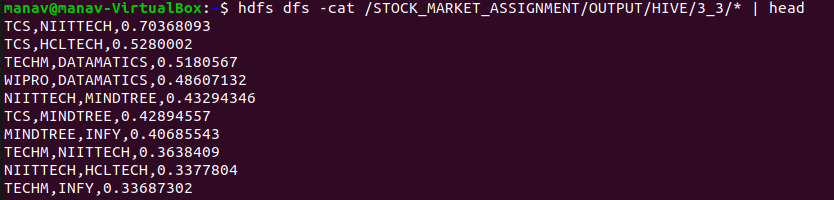
FIELDS TERMINATED BY ','

STORED AS TEXTFILE

SELECT \* FROM UC2\_STOCK.PEARSONCORITSTOCK;

##### G:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\HIVE_3_1.PNGOutput:





### Pig

##### Pig Commands:

FILTER\_DATA = FILTER STOCK BY TOTTRDQTY >= 300000 AND

SUBSTRING(TIMESTAMP,0,4) == '2017';

LIMIT\_DATA = LIMIT FILTER\_DATA 25;

STORE LIMIT\_DATA INTO 'hdfs://localhost:9000/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/PIG/3\_1/'

USING PigStorage (',');

ITSTOCK = FILTER FILTER\_DATA BY SYMBOL IN

('HCLTECH', 'NIITTECH', 'TATAELXSI','TCS', 'INFY', 'WIPRO', 'DATAMATICS', 'TECHM','MINDTREE', 'OFSS');

STORE ITSTOCK INTO 'hdfs://localhost:9000/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/PIG/3\_2/'

USING PigStorage (',');

ITSTOCK1 = FILTER FILTER\_DATA BY SYMBOL IN

('HCLTECH', 'NIITTECH', 'TATAELXSI','TCS', 'INFY', 'WIPRO', 'DATAMATICS',

'TECHM','MINDTREE', 'OFSS');

ITSTOCK2 = FILTER FILTER\_DATA BY SYMBOL IN ('HCLTECH', 'NIITTECH', 'TATAELXSI','TCS', 'INFY', 'WIPRO', 'DATAMATICS','TECHM','MINDTREE', 'OFSS');

ITSTOCK\_1\_2 = JOIN ITSTOCK1 BY $10, ITSTOCK2 BY $10;

FLTR = FILTER ITSTOCK\_1\_2 BY $0 > $13;

GROUP\_SYMBOL = GROUP FLTR BY ($0, $13);

CORITSTOCK = FOREACH GROUP\_SYMBOL {

CNT = COUNT(FLTR.$0);

SC = FOREACH FLTR GENERATE $5 AS (CLOSE1:DOUBLE), $18 AS (CLOSE2:DOUBLE),

$5 \* $5 AS (CC1:DOUBLE), $18 \* $18 AS (CC2:DOUBLE), $5 \* $18 AS (C1C2:DOUBLE);

GENERATE group.$0 AS (SYMBOL1:CHARARRAY), group.$1 AS (SYMBOL2:CHARARRAY),

SUM(SC.CLOSE1) AS (SUM1:DOUBLE), SUM(SC.CLOSE2) AS (SUM2:DOUBLE),

SUM(SC.CC1) AS (SUM1SQ:DOUBLE), SUM(SC.CC2) AS (SUM2SQ:DOUBLE),

SUM(SC.C1C2) AS (PSUM:DOUBLE), CNT \* 1.0 AS (N:DOUBLE), FLTR.$10 AS T;

};

CORITSTOCK\_SORT = ORDER CORITSTOCK BY SYMBOL1, SYMBOL2, T;

PEARSONCORITSTOCK = FOREACH CORITSTOCK GENERATE SYMBOL1, SYMBOL2,

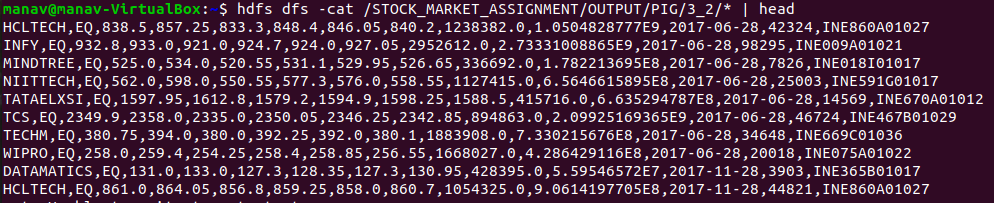
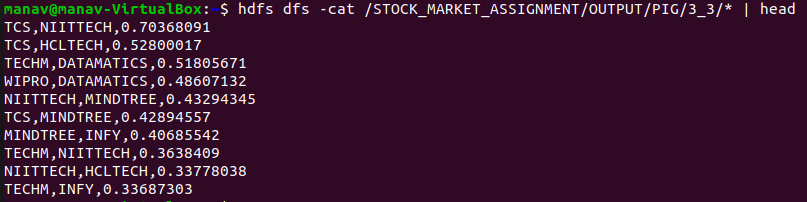
ROUND\_TO((PSUM - (SUM1 \* SUM2 / N)) / SQRT((SUM1SQ - (SUM1 \* SUM1) / N) \* (SUM2SQ - (SUM2 \* SUM2) / N)),8) AS (PEARSONCOEFFICIENT:DOUBLE);

PEARSONCORITSTOCK\_SORT = ORDER PEARSONCORITSTOCK BY PEARSONCOEFFICIENT DESC;

STORE PEARSONCORITSTOCK\_SORT INTO 'hdfs://localhost:9000/STOCK\_MARKET\_ASSIGNMENT/OUTPUT/PIG/3\_3/'

USING PigStorage (',');

##### G:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\PIG_3_1.PNGOutput:



### Spark

##### Spark Commands:

val STOCK = sc.textFile(InputFile).map(row => row.split(","))

val FilterMap = STOCK.filter(col => col(TOTTRDQTY).toLong >= 300000 && col(TIMESTAMP).toString.split("-")(0) == "2017").map(col => {(col(SYMBOL).toString, col(SERIES).toString, col(OPEN).toDouble, col(HIGH).toDouble, col(LOW).toDouble, col(CLOSE).toDouble, col(LAST).toDouble, col(PREVCLOSE).toDouble, col(TOTTRDQTY).toDouble, col(TOTTRDVAL).toDouble, col(TIMESTAMP).toString, col(TOTALTRADES).toLong, col(ISIN).toString)})

val FilterMap25 = sc.parallelize(FilterMap.take(25))

FilterMap25.map(r => r.productIterator.mkString(",")).saveAsTextFile(OutputFile+"3\_1")

val ITSTOCK = FilterMap.filter(col => col.\_1 == "HCLTECH" || col.\_1 == "NIITTECH" || col.\_1 == "TATAELXSI" || col.\_1 == "TCS" || col.\_1 == "INFY" || col.\_1 == "WIPRO" || col.\_1 == "DATAMATICS" || col.\_1 == "TECHM" || col.\_1 == "MINDTREE" || col.\_1 == "OFSS")

ITSTOCK.map(r => r.productIterator.mkString(",")).saveAsTextFile(OutputFile+"3\_2")

val ITSTOCKA = ITSTOCK.map(col => {(col.\_11, col.\_1, col.\_6)}).map(col => (col.\_1, {(col.\_2, col.\_3)}))

val joined=ITSTOCKA.join(ITSTOCKA)

val fltr\_join = joined.filter(col => col.\_2.\_1.\_1 > col.\_2.\_2.\_1).sortBy(\_.\_2.\_1.\_1).sortBy(\_.\_2.\_2.\_1).sortBy(\_.\_1)

val fltr\_map = fltr\_join.map(col => col.\_2)

val fltr\_map2 = fltr\_map.map(col => ((col.\_1.\_1, col.\_2.\_1), (col.\_1.\_2, col.\_2.\_2, col.\_1.\_2\*col.\_2.\_2)))

val grp = fltr\_map2.groupByKey()

implicit def iterebleWithAvg(data:Iterable[Double]) = new {

def avg:Double = data.sum / data.size

}

implicit def iterebleWithStdDev(data:Iterable[Double]) = new {

def stddev:Double = org.apache.spark.util.StatCounter(data).stdev

}

val grp\_map = grp.map(col => {((col.\_1.\_1, col.\_1.\_2), ((col.\_2.map(y=>y.\_3).avg - col.\_2.map(y=>y.\_1).avg \* col.\_2.map(y=>y.\_2).avg)/(col.\_2.map(y=>y.\_1).stddev \* col.\_2.map(y=>y.\_2).stddev)))})

val pearsoncoefficient = grp\_map.map(col => (col.\_1.\_1, col.\_1.\_2, col.\_2.toDouble)).sortBy(\_.\_3, false)

pearsoncoefficient.map(r => r.productIterator.mkString(","))

.saveAsTextFile(OutputFile+"3\_3")

##### G:\PGD_class\STOCK_MARKET_ASSIGNMENT\OUTPUT\SPARK_3_1.PNGOutput:

