A PROJECT ON ONLINE RETAIL STORE 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

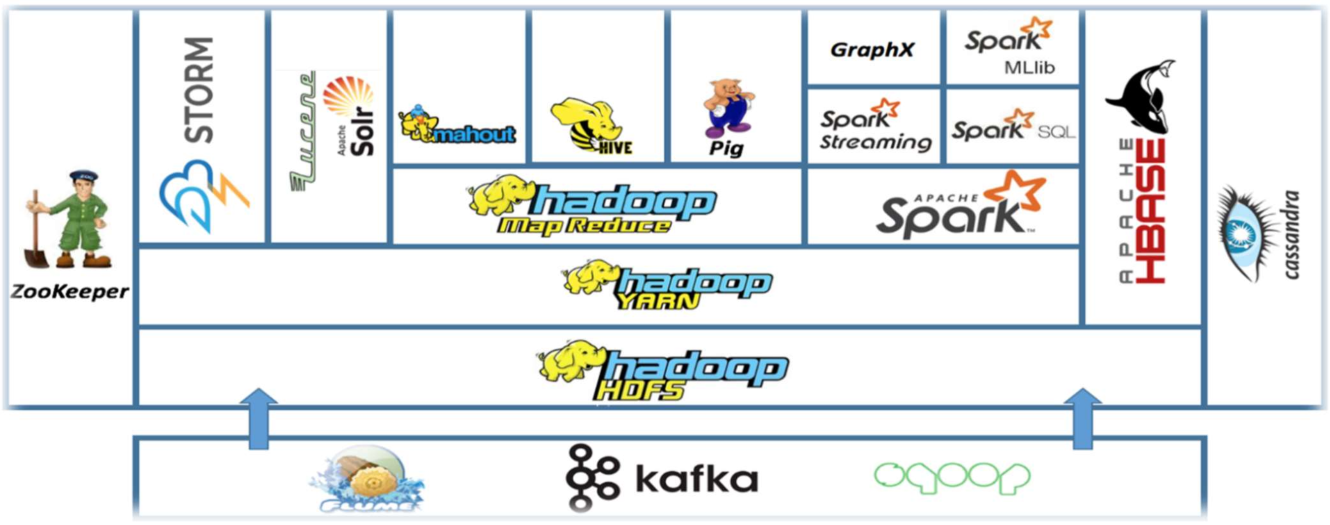
Why Hadoop?

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



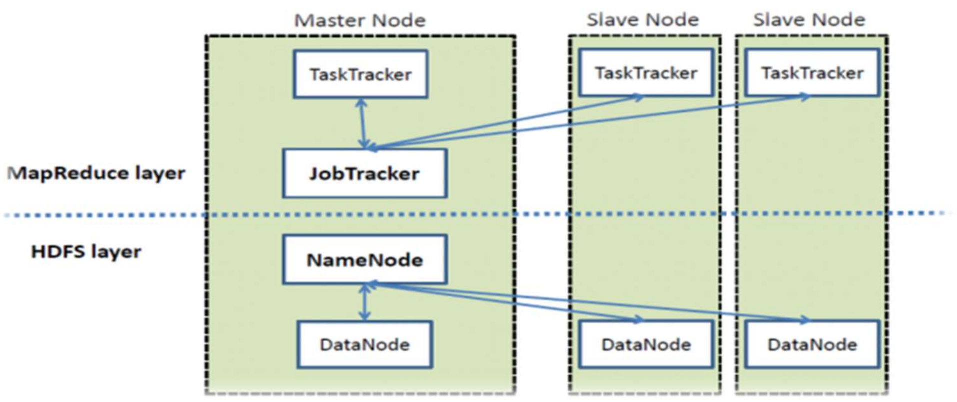
All the modules in Hadoop are designed with a fundamental assumption that hardware failures (of individual machines, or racks of machines) are common and thus should be automatically handled in software by the framework. Apache Hadoop's MapReduce and HDFS components originally derived respectively from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

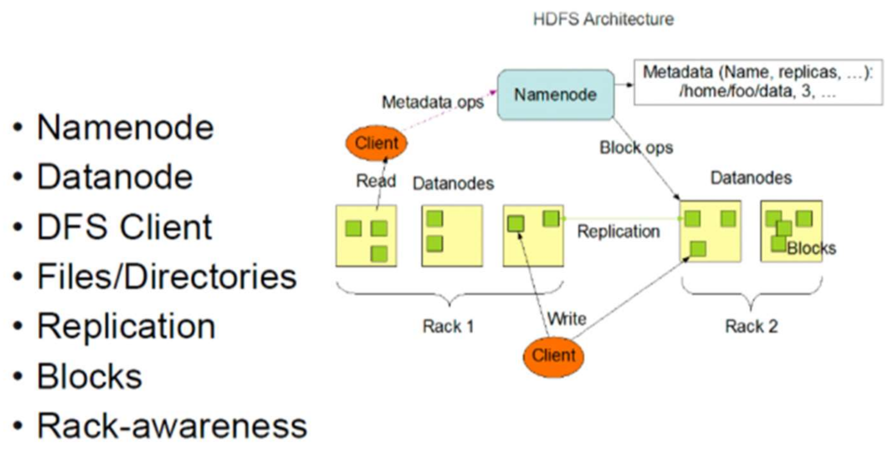
There are two primary components at the core of Apache Hadoop 1.x: the Hadoop Distributed File System (HDFS) and the MapReduce parallel processing framework. These are both open source projects, inspired by technologies created inside Google.

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:

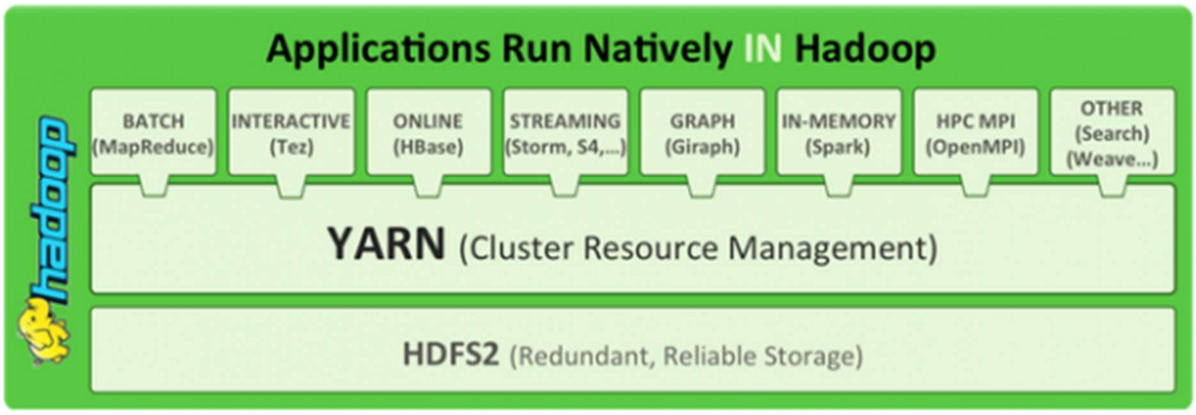
Scalability: The processing power in data centers continues to grow quickly. Because YARN ResourceManager focuses exclusively on scheduling, it can manage those larger clusters much more easily.

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. Revenue Aggregate by Country for top 5 countries
2. Sales Metrics like NumCustomers, NumTransactions, AvgNumItems, MinAmtperCustomer, MaxAmtperCustomer, AvgAmtperCustomer, StdDevAmtperCustomeretc. .. by country for top 5 countries
3. Daily sales activity like numvisits, totalamount monthly and quarterly for 1 year.
4. Hourly sales activity like numvisits, totalamount per hour of day.
5. Basket size distribution (Note: Basket size = number of items in a transaction) (in this questions, we would like to know that, number of transactions by each basket size i.e. number of transactions with 3 size, number of transactions with 4 size etc.
6. Top 20 Items sold by frequency
7. Customer Lifetime Value distribution by intervals of 1000's (Customer Life time Value = total spend by customer in his/her tenure with the company) (In this question, we would like to calculate how many customers with CLV between 1-1000, 1000-2000 etc.). Please note that we don't want calculate bins manually and it required to create bins dynamically.

Data definitions:

- *InvoiceNo-integer -Transaction Number
- *StockCode-character -SKU Code (Product Code)
- *Description -character -Product Description
- *Quantity -int-Quantity ordered
- *InvoiceDate-character -Transaction Data
- *UnitPrice-float-Price per unit quantity
- *CustomerID-character -Customer ID *Country -character -Customer location

Pre-information: online retail dataset containing transactions occurring between 01/12/2010 and 09/12/2011 for a UK-based and registered non-store online retail

Codes and Workflows

Question 1

Python Mapper

```
#!usr/bin/python

 #
 #   Analysis 1:
 #      Mapper for isolating components of Revenue Aggregate *
 #          by country for top 5 countries  **
 #  start of code
 import sys
 for line in sys.stdin:
    row=line.strip()    # removeing white spaces
    components=row.split("\t")      # separating individual components
    try:
       if ( "" not in components and int(components[3])>= 0 and float(components[5])>= 0.0 ):   #
 filtration -> quality control
          print('%s\t%f'%(components[7],int(components[3])*float(components[5])))   # key: country ;
 values : quantity and unit price
       else:
          continue   #ignore condition failures -> quality control
    except Exception:
       continue    # ignore filtration failure -> quality control

 # end of code
 # results mentioned in the reducer
```

Python Reducer

```
#!usr/bin/python
#
#   Analysis 1:
#      Reducer for calculating value of Revenue Aggregate *
#          by country for top 5 countries  *
#              from mapper response       **

# ** start of code
```

```
import sys
from operator import itemgetter
data_list=[]        # local list
for line in sys.stdin:
    data_line=line.strip()      #remove whitespaces
    elements=data_line.split("\t")  #segregate components
    data_list.append(elements)  # generate local list
if len(data_list) is 0:
    print(" Input Stream Error ... -> Reducer didn't receive any data ")
    sys.exit(1)        # fatal error due to reducer not receiving any data
data_dict={}  # storing all revenue aggregates
for elem in data_list:
    try:
        # test code for input errors
        if elem[0] not in data_dict:
            # new found unique key :: country
            data_dict[elem[0]]=float(elem[1])
        else:
            #  already available country key
            data_dict[elem[0]] = data_dict[elem[0]]+float(elem[1])

    except Exception:
        print (" Input Data Error ... -> Dictionary not generated or In-accessable data list ")
        sys.exit(1)

#helping function
def maxElem(a={}):
    #function to find out maximum valued key
    key=0 #raw initialization
    val=0
    for i in a:
        if a[i] > val: #is greater than existing check
            key=i
            val=a[i]
    # returning result
    return key
    #end of function

# finding top 5
data_dict_wk=data_dict  # duplicating the dictonary to avoid original data distortion
analysis1=[] # list to contain the results of the analysis1
for i in range(5):
    t=maxElem(data_dict_wk)     # max value containg key of the dictionary
    try:
        # test removability of hashed data component
        val=data_dict_wk.pop(t)  # releasing the k,v pair from the disctionary
    except Exception:
        print ("  Data Error:: --> Inadequate availability  ")
        sys.exit(1)

    analysis1.append('%s\t%f'%(t,val))
    print('%s\t%f'%(t,val))

# end of job for analysis1

# ** end of code

##
# operation using linux core ->
# command tested for a modified input file of 1000 datapoints from the original OnlineRetail.txt
# linux core command
#hdfs dfs -cat /assign1/OnlRet_1K.txt | python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/analysis1map.py | sort| shuf  | python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/analysis1red.py
# results obtained
#**..
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-10-22 03:28:09,071 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
#United Kingdom  23099.540000
#France  855.860000
#Australia       358.250000
#Netherlands     192.600000
# Data Error:: --> Inadequate availability
#kali@kali:~$
#**..
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign1.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign1.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign1.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign1.py" -input /assign1/OnlineRetail.txt -output
/assign1/pythonMR_jobs/analysis1pmr
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis1pmr/part*
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-10-26 21:18:02,723 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
#United Kingdom  7308391.554004
#Netherlands     285446.340000
#EIRE    265545.900000
#Germany 228867.140000
#France  209024.050000
```

```
#kali@kali:~$
#**..
```

Hive sql code

```
-- Analysis 1:

 -- Revenue Aggregate by country for top 5 countries

 -- start of codes
 -- one time jobs
 -- **## Please avoid lines here on forward if retaildb is available in 'show databases' command
 and contains data                 ##**
 -- **## if avoiding these lines till line 23 directly jump to codes in analysis section of the code
 which next to this section     ##**
 create database RetailDB;
 use RetailDB;
 -- table creation
 create table data_raw_headless (InvoiceNo string,StockCode string,Description string,Quantity
 int,InvoiceDate string,UnitPrice float,CustomerID string,Country string) row format delimited fields
 terminated by '\t' lines terminated by '\n' tblproperties("skip.header.line.count"="1");
 load data inpath 'hdfs://localhost:9000/user/hive/warehouse/OnlineRetail.txt' into table
 data_raw_headless;
 --check number of datarows
 select count (stockcode) from data_raw_headless1 where stockcode!='';
 -- result 541909
 --cleaning of debris
 create table data_cleaned as select * from data_raw_headless1
 where (quantity>0 AND unitprice>=0.0 AND customerid != '');
 --confirmation and count of cleaned data
 show tables;
 select count (stockcode) from data_cleaned where stockcode!='';
 -- result 397924
 -- cleaned of debris

 -- analysis job
 create table kv_anlysis1(country string,totalcost float);  -- table for calculation of aggregate
 insert into kv_anlysis1  select (country),(quantity*unitprice) from data_cleaned; -- population of
 table
 create table analysis1Result (country string,revAggr float); -- table reveneue aggregate of all
 countries
 insert into analysis1Result select country,sum(totalcost) as revenue from kv_asses11 group by
 country order by revenue desc; --populating the table ordered
 create table analysis1T5 row format delimited fields terminated by '\t' stored as textfile as select *
 from analysis1Result order by revaggr desc limit 5; -- final required result;

 -- results obtained
 --** HIVE shell
 --
```

```
--hive> select * from  analysis1T5;
--OK
--United Kingdom  7308391.5
--Netherlands     285446.34
--EIRE    265545.9
--Germany 228867.14
--France  209024.05
--Time taken: 0.211 seconds, Fetched: 5 row(s)
--hive>
--**
--
-- **HDFS CORE
--kali@kali:~$ hdfs dfs -cat /user/hive/warehouse/retaildb.db/analysis1t5/*
--Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
--2020-10-29 14:11:11,990 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
--United Kingdom  7308391.5
--Netherlands     285446.34
--EIRE    265545.9
--Germany 228867.14
--France  209024.05
--kali@kali:~$
--
```

Pig Code

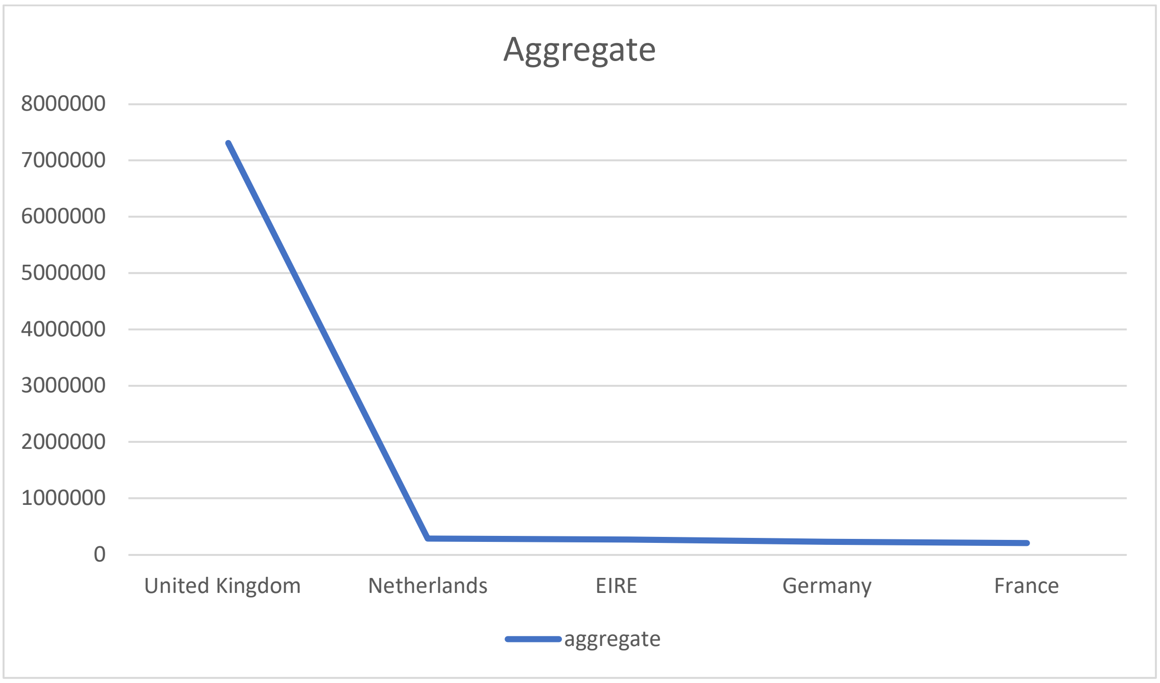
```
/*Analysis 1:

  Revenue Aggregate by country for top 5 countries */

  -- start of code

  --raw data load with defined schema
  data_raw= LOAD '/home/kali/Hadoop/Local_Datasets/OnlineRetail.txt' USING PigStorage() as
  (InvoiceNo:chararray,StockCode:chararray,Description:chararray,Quantity:int,InvoiceDate:Datetime,UnitP
  rice:float,CustomerID:chararray,Country:chararray);
  data_cleaned = FILTER data_raw BY (Quantity>=0 AND UnitPrice>=0.0 AND CustomerID!=''); --cleaning
  with condition
  ctrRawG= GROUP data_raw ALL;  -- grouping raw to count
  ctrClnG= GROUP data_cleaned ALL; -- grouping cleaned to count
  ctrRaw= FOREACH ctrRawG GENERATE COUNT(data_raw.Quantity);  --generating count raw
  ctrCln= FOREACH ctrClnG GENERATE COUNT(data_cleaned.Quantity); -- generating count cleaned
  -- dumping to trigger results' calculation
  dump ctrRaw --value :: (541909)
  dump ctrCln --value :: (397924)
```

```
-- cleaned of debris in data

--analysis job
kvp_asgn1 = FOREACH data_cleaned GENERATE Country as (Country:chararray), UnitPrice*Quantity as
(TotalCost:float); -- kv map for generating aggregate
resGrp = GROUP kvp_asgn1 BY Country; -- group by country
--illustrate -- check type for next line
resFinal = FOREACH resGrp GENERATE group as (Country:chararray),SUM(kvp_asgn1.TotalCost) as
(Revenue:float); --group sum of revenues
ordered_res = ORDER resFinal BY Revenue DESC; -- descending order
ordered_res5= LIMIT ordered_res 5; -- top 5 countries -> revenue aggregate
STORE ordered_res5 INTO '/home/kali/Hadoop/Results/pig_results/analysis1/' USING PigStorage();

/*
** Results obtained
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis1/part*
United Kingdom  7308391.5
Netherlands     285446.34
EIRE    265545.9
Germany 228867.14
France  209024.05
kali@kali:~$
*/
```

Spark Code

```
// Spark code for Analysis 1: Revenue Aggregate by country
//Start of Code
//cleaning of visible debris
var data_raw= sc.textFile("hdfs://localhost:9000/assign1/OnlineRetail.txt")
var data_raw_split= data_raw.map(x=>x.split("\t"))
var data_headless=data_raw_split.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1) else iter }
data_headless.count
var data_cleaned=data_headless.filter{x=> if((x(3).toInt >=0 ) && (x(5).toFloat >=0.0) && (x(6)!=""))true
else false}
data_cleaned.count
// cleaned data of primary debris

// analysis job
var kvp_Asgn1=data_cleaned.map(x=> {(x(7),x(3).toInt,x(5).toFloat)})  // kv map for intake checking
kvp_Asgn1.take(5) //check kv maps
var kvp_final_Asgn1=data_cleaned.map(x=> {(x(7), x(3).toInt * x(5).toFloat)}) //final kv map for calculating
aggregate
kvp_final_Asgn1.take(5) // checking multiplier value to kvp_Asgn1
var results=kvp_final_Asgn1.reduceByKey((i,j)=>(i+j)) //Revenue
var res_sorted= results.sortBy(_._2,false) //arrange in descending order
sc.parallelize(res_sorted.take(5)).saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis1")
```

```
//end of code

// Solution obtained
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis1/part-*
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-10-18 17:43:38,680 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
//(United Kingdom,7309325.0)
//(Netherlands,285446.5)
//(EIRE,265546.38)
//(Germany,228866.53)
//(France,209023.77)
//kali@kali:~$
```



Question 2

Python Mapper

```
#!usr/bin/python
#
# *******
# **     Analysis 2:
# *    Mapper for calculating the results of                                    **
# *   Sales Metrics like NumCustomers, NumTransactions, AvgNumItems,                        **
# *    MinAmtperCustomer, MaxAmtperCustomer, AvgAmtperCustomer.                         **
# *     by country for top 5 countries                                        **
# *******
# start of code
import sys
for line in sys.stdin:
    row=line.strip()    # removeing white spaces
    components=row.split("\t")         # separating individual components
    try:
        if ( "" not in components and int(components[3])>= 0 and float(components[5])>= 0.0 ):      #
filtration -> quality control
            # generating K,V pairs after filtering out debris in data

print('%s\t%s\t%d\t%f\t%s'%(components[7],components[0],int(components[3]),float(components[5]),
components[6]))
        else:
           #print(components)    # debug lines for error testing in code
           continue           #ignore condition failures -> quality control
    except Exception:
        #print(components)      # debug lines for error testing in code
        continue              #ignore condition failures -> quality control

# end of code
```

Python Reducers

1.

```
#!usr/bin/python
# *******
# **    Analysis 2:
# *    Reducer for calculating the results of                                    **
# *   Sales Metrics like NumCustomers, NumTransactions, AvgNumItems,                        **
# *    MinAmtperCustomer, MaxAmtperCustomer, AvgAmtperCustomer.                          **
# *     by country for top 5 countries                                        **
# *******
# **** Specific operation:  Sales Metric of Number of Customers by country for top 5 ****
# start of code
import sys
from operator import itemgetter
data_list=[]
for line in sys.stdin:
    data_line=line.strip()
    elements=data_line.split("\t")
    data_list.append(elements)  # generate local list
if len(data_list) == 0:
    print(" Input Stream Error ... -> Reducer didn't receive any data ")
    sys.exit(1)
```

```
data_dict={}  # storing dictionary for all primary k-v sets
for elem in data_list:
    try:
        # test code for input errors
        if elem[0] not in data_dict:
            # new found unique key :: country
            data_dict[elem[0]]=[elem[4]]
        else:
            #  already available country key
            data_dict[elem[0]].append(elem[4])

    except Exception:
         print (" Input Data Error ... -> Dictionary not generated or Inaccessible data list ")
         sys.exit(1)
#@print data_list       # debugging test code
data_dict_res={}
for member in data_dict:
    data_dict_res[member]=len(set(data_dict[member]))



#helping function
def maxElem(a={}):
    #function to find out maximum valued key
    key=0 #raw initialization
    val=0
    for i in a:
        if a[i] > val:
            key=i
            val=a[i]
    # yielding results
    return key
    #end of function

# finding top 5
data_dict_wk=data_dict_res
analysis2x1=[]
for i in range(5):
    t=maxElem(data_dict_wk)
    #@print t
    #@print data_dict_wk[t]
    try:
        # test removability of hashed data component
        val=data_dict_wk.pop(t)
    except Exception:
        print (" Data Error:: --> Inadequate availability  ")
        sys.exit(1)

    analysis2x1.append('%s\t%f'%(t,val))
    print('%s\t%d'%(t,val))

# end of job for analysis2 job1

#@ are test lines for manual debugging codes



# ** end of code

##
# operation using linux core ->
# command tested for a modified input file of 1000 datapoints from the original OnlineRetail.txt
# linux core command
# results obtained with command
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/OnlRet_1K.txt | python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py|sort|shuf| python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex1.py
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-10-30 12:24:54,756 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
#United Kingdom  43
#Australia       1
#Netherlands     1
#France  1
# Data Error:: --> Inadequate availability
#kali@kali:~$
#**..
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex1.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex1.py" -input /assign1/OnlineRetail.txt -
output /assign1/pythonMR_jobs/analysis2pmr/Job1
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis2pmr/Job1/part*
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-10-30 13:57:31,036 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
#United Kingdom  3921
#Germany 94
#France  87
#Spain   30
```

```
#Belgium 25
#kali@kali:~$
#**..
```

2.

```
#!usr/bin/python

# *******
# **    Analysis 2:
# *    Reducer for calculating the results of                                        **
# *   Sales Metrics like NumCustomers, NumTransactions, AvgNumItems,                        **
# *    MinAmtperCustomer, MaxAmtperCustomer, AvgAmtperCustomer.                               **
# *     by country for top 5 countries                                          **
# *******
# **** Specific operation:  Sales Metric of Number of Transactions by country for top 5 ****
# start of code
import sys
from operator import itemgetter
data_list=[]
for line in sys.stdin:
   data_line=line.strip()
   elements=data_line.split("\t")
   data_list.append(elements)  # generate local list
if len(data_list) == 0:
   print(" Input Stream Error ... -> Reducer didn't receive any data ")
   sys.exit(1)
data_dict={}  # storing dictionary for all primary k-v sets
for elem in data_list:
   try:
      # test code for input errors
      if elem[0] not in data_dict:
         # new found unique key :: country
         data_dict[elem[0]]=[elem[1]]
      else:
         #  already available country key
         data_dict[elem[0]].append(elem[1])

   except Exception:
       print (" Input Data Error ... -> Dictionary not generated or Inaccessible data list ")
       sys.exit(1)
#@print data_list       # debugging test code
data_dict_res={}
for member in data_dict:
   data_dict_res[member]=len(set(data_dict[member]))


#helping function
def maxElem(a={}):
    #function to find out maximum valued key
    key=0 #raw initialization
    val=0
    for i in a:
        if a[i] > val:
            key=i
            val=a[i]
    # yielding results
    return key
    #end of function

# finding top 5
data_dict_wk=data_dict_res
analysis2x2=[]
for i in range(5):
    t=maxElem(data_dict_wk)
    #@print t
    #@print data_dict_wk[t]
    try:
        # test removability of hashed data component
        val=data_dict_wk.pop(t)
    except Exception:
        print (" Data Error:: --> Inadequate availability  ")
        sys.exit(1)

    analysis2x2.append('%s\t%f'%(t,val))
    print('%s\t%d'%(t,val))

# end of job for analysis2 job2

#@ are test lines for manual debugging codes



# ** end of code

##
# operation using linux core ->
# command tested for a modified input file of 1000 datapoints from the original OnlineRetail.txt
# linux core command
# results obtained with command
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/OnlRet_1K.txt | python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py|sort|shuf| python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex2.py
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-10-30 18:29:23,656 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
```

```
#United Kingdom  58
#Australia       1
#Netherlands     1
#France  1
# Data Error:: --> Inadequate availability
#kali@kali:~$
#**..
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex2.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex2.py" -input /assign1/OnlineRetail.txt -
output /assign1/pythonMR_jobs/analysis2pmr/Job2
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis2pmr/Job2/part*
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-10-30 18:27:30,430 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
#United Kingdom  16649
#Germany 457
#France  389
#EIRE    260
#Belgium 98
#kali@kali:~$
#**..
```

3.

```
#!usr/bin/python
#
# *******
# **    Analysis 2:
# *    Reducer for calculating the results of                                    **
# *   Sales Metrics like NumCustomers, NumTransactions, AvgNumItems,                        **
# *    MinAmtperCustomer, MaxAmtperCustomer, AvgAmtperCustomer.                     **
# *     by country for top 5 countries                                  **
# *******
# **** Specific operation:  Sales Metric of Average Number of Items by country for top 5 ****
# start of code
import sys
from operator import itemgetter
data_list=[]
for line in sys.stdin:
    data_line=line.strip()
    elements=data_line.split("\t")
    data_list.append(elements)  # generate local list
if len(data_list) == 0:
    print(" Input Stream Error ... -> Reducer didn't receive any data ")
    sys.exit(1)
data_dict={}  # storing dictionary for all primary k-v sets
for elem in data_list:
    try:
        # test code for input errors
        if elem[0] not in data_dict:
            # new found unique key :: country
            data_dict[elem[0]]=int(elem[2])
        else:
            #  already available country key
            data_dict[elem[0]]=data_dict[elem[0]]+int(elem[2])

    except Exception:
         print (" Input Data Error ... -> Dictionary not generated or Inaccessible data list ")
         sys.exit(1)
#@print data_list       # debugging test code
data_dict_res={}
tcountry =len(data_dict)
for member in data_dict:
    data_dict_res[member]=data_dict[member]*1.0/tcountry


#helping function
def maxElem(a={}):
    #function to find out maximum valued key
    key=0 #raw initialization
    val=0
    for i in a:
        if a[i] > val:
            key=i
            val=a[i]
    # yielding results
    return key
    #end of function

# finding top 5
data_dict_wk=data_dict_res
analysis2x3=[]
for i in range(5):
    t=maxElem(data_dict_wk)
    #@print t
    #@print data_dict_wk[t]
    try:
        # test removability of hashed data component
        val=data_dict_wk.pop(t)
    except Exception:
        print (" Data Error:: --> Inadequate availability  ")
        sys.exit(1)

    analysis2x3.append('%s\t%f'%(t,val))
    print('%s\t%f'%(t,val))

# end of job for analysis2 job3

#@ are test lines for manual debugging codes



# ** end of code

##
# operation using linux core ->
# command tested for a modified input file of 1000 datapoints from the original OnlineRetail.txt
# linux core command
# results obtained with command
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/OnlRet_1K.txt | python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py|sort|shuf| python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex3.py
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-10-30 21:19:03,173 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
#United Kingdom  3054.000000
#France  112.250000
#Australia       26.750000
#Netherlands     24.250000
# Data Error:: --> Inadequate availability
#kali@kali:~$
##**..
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex3.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex3.py" -input /assign1/OnlineRetail.txt -
output /assign1/pythonMR_jobs/analysis2pmr/Job3
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis2pmr/Job3/part*
```

```
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-10-30 21:25:41,756 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
#United Kingdom  115391.135135
#Netherlands     5430.729730
#EIRE    3797.972973
#Germany 3223.324324
#France  3012.756757
#kali@kali:~$
#**..
```

4.

```
#!usr/bin/python
#
# **    Analysis 2:
# *    Reducer for calculating the results of                                  **
# *   Sales Metrics like NumCustomers, NumTransactions, AvgNumItems,                       **
# *    MinAmtperCustomer, MaxAmtperCustomer, AvgAmtperCustomer.                           **
# *     by country for top 5 countries                                         **
# *******
# **** Specific operation:  Sales Metric of Minimum Amount Spent Per Customer ****
# start of code
import sys
from operator import itemgetter
data_list=[]
for line in sys.stdin:
   data_line=line.strip()
   elements=data_line.split("\t")
   data_list.append(elements)  # generate local list
if len(data_list) == 0:
   print(" Input Stream Error ... -> Reducer didn't receive any data ")
   sys.exit(1)

data_dict={}  # storing dictionary for all primary k-v sets
for elem in data_list:
   try:
      # test code for input errors
      if elem[4] not in data_dict:
         # new customer found
         data_dict[elem[4]]={}
         data_dict[elem[4]][elem[1]]=int(elem[2])*float(elem[3])
      else:
         # customer exists
         if elem[1] in data_dict[elem[4]]:
            # invoice of the customer exists -> adding to the sum
            data_dict[elem[4]][elem[1]]= data_dict[elem[4]][elem[1]]+int(elem[2])*float(elem[3])
         else:
            # invoice doesn't exists for the customer -> generating new invoice info
            data_dict[elem[4]][elem[1]]=int(elem[2])*float(elem[3])
    except Exception:
        print (" Input Data Error 1... -> Dictionary not generated or Inaccessible data list ")
        sys.exit(1)
#@print data_list       # debugging test code
#@print data_dict
resultsMIN={}       # dictionary for final tasks but dataset is expendable
# segregating final informatics
for customer in data_dict:
    # creating a workable dictionary for further Jobs
    resultsMIN[customer]=min(data_dict[customer].values())

resultsErosiv = resultsMIN # erosive dictionary for sort and print

#helping function
def maxElem(a={}):
    #function to find out maximum valued key
    key=a.keys()[0] #raw initialization
    val=0
    for i in a:
        if a[i] > val:
            key=i
            val=a[i]
    # yielding results
    return key
    #end of function

# sort and print using erosive dictionary
steps=len(resultsErosiv)
for i in range(steps):
    t=maxElem(resultsErosiv)
    #@print t
    #@print resultsErosiv[t]
    try:
        # test removability of hashed data component
        val=resultsErosiv.pop(t)
    except Exception:
        print (" Data Error:: --> Inadequate availability  ")
        sys.exit(1)



    print('%s\t%f'%(t,val))
#
# end of job for analysis2 job4

#@ are test lines for manual debugging codes

# end of code
```

```
##
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex4.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex4.py" -input /assign1/OnlineRetail.txt -
output /assign1/pythonMR_jobs/analysis2pmr/Job4
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis2pmr/Job4/part*|head -n 10
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-11-08 11:20:04,268 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
#12346   77183.600000
#15749   7837.500000
#12357   6207.670000
#12688   4873.810000
#12752   4366.780000
#18251   4314.720000
#12536   4161.060000
#12378   4008.620000
#15195   3861.000000
#12435   3850.900000
#cat: Unable to write to output stream.
#kali@kali:~$
#**..
#*** note that this job limited to 10 outputs from the head of the file systems in storage to avoid
printing 4339 records cluttering the shell screen
# ** this restriction imposed to limit output generated a cat error from the resulting stream
```

5.
```
#!usr/bin/python
#
# *******
# **    Analysis 2:
# *    Reducer for calculating the results of                              **
# *   Sales Metrics like NumCustomers, NumTransactions, AvgNumItems,
**
# *    MinAmtperCustomer, MaxAmtperCustomer, AvgAmtperCustomer.
**
# *     by country for top 5 countries                                  **
# *******
# **** Specific operation:  Sales Metric of Maximum Amount Spent Per Customer ****
# start of code
```

```
import sys
from operator import itemgetter
data_list=[]
for line in sys.stdin:
    data_line=line.strip()
    elements=data_line.split("\t")
    data_list.append(elements)  # generate local list
if len(data_list) == 0:
    print(" Input Stream Error ... -> Reducer didn't receive any data ")
    sys.exit(1)
data_dict={}  # storing dictionary for all primary k-v sets
for elem in data_list:
    try:
        # test code for input errors
        if elem[4] not in data_dict:
            # new customer found
            data_dict[elem[4]]={}
            data_dict[elem[4]][elem[1]]=int(elem[2])*float(elem[3])
        else:
            # customer exists
            if elem[1] in data_dict[elem[4]]:
                # invoice of the customer exists -> adding to the sum
                data_dict[elem[4]][elem[1]]=
data_dict[elem[4]][elem[1]]+int(elem[2])*float(elem[3])
            else:
                # invoice doesn't exists for the customer -> generating new invoice info
                data_dict[elem[4]][elem[1]]=int(elem[2])*float(elem[3])
    except Exception:
         print (" Input Data Error ... -> Dictionary not generated or Inaccessible data list ")
         sys.exit(1)
#@print data_list       # debugging test code
resultsMAX={}       # dictionary for final tasks but dataset is expendable
# segregating final informatics
for customer in data_dict:
    # creating a workable dictionary for further Jobs
    resultsMAX[customer]=max(data_dict[customer].values())

resultsErosiv = resultsMAX # erosive dictionary for sort and print
#helping function
def maxElem(a={}):
    #function to find out maximum valued key
    key=a.keys()[0] #raw initialization
    val=0
    for i in a:
        if a[i] > val:
            key=i
            val=a[i]
    # yielding results
    return key
    #end of function

# sort and print using erosive dictionary
steps=len(resultsErosiv)
for i in range(steps):
    t=maxElem(resultsErosiv)
    #@print t
    #@print resultsErosiv[t]
    try:
        # test removability of hashed data component
        val=resultsErosiv.pop(t)
    except Exception:
        print (" Data Error:: --> Inadequate availability  ")
        sys.exit(1)

    print('%s\t%f'%(t,val))
#
# end of job for analysis2 job5

#@ are test lines for manual debugging codes

# end of code

##
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex5.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex5.py" -input
/assign1/OnlineRetail.txt -output /assign1/pythonMR_jobs/analysis2pmr/Job5
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis2pmr/Job5/part*|head -n 10
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-11-08 11:20:15,499 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
#16446   168469.600000
#12346   77183.600000
#15098   38970.000000
#17450   31698.160000
#12415   22775.930000
#18102   22206.000000
#15749   21535.900000
#14646   20277.920000
```

```
#12931   18841.480000
#14156   16774.720000
#cat: Unable to write to output stream.
#kali@kali:~$
#**..
#*** note that this job limited to 10 outputs from the head of the file systems in storage
to avoid printing 4339 records cluttering the shell screen
# ** this restriction imposed to limit output generated a cat error from the resulting
stream
```

6.

```
#!usr/bin/python
#
# *******
# **    Analysis 2:
# *    Reducer for calculating the results of                              **
# *   Sales Metrics like NumCustomers, NumTransactions, AvgNumItems,
**
# *    MinAmtperCustomer, MaxAmtperCustomer, AvgAmtperCustomer.
**
# *     by country for top 5 countries                                     **
# *******
# **** Specific operation:  Sales Metric of Average Amount Spent Per Customer ****
# start of code
import sys
from operator import itemgetter
data_list=[]
for line in sys.stdin:
    data_line=line.strip()
    elements=data_line.split("\t")
    data_list.append(elements)  # generate local list
if len(data_list) == 0:
    print(" Input Stream Error ... -> Reducer didn't receive any data ")
    sys.exit(1)
data_dict={}  # storing dictionary for all primary k-v sets
for elem in data_list:
    try:
        # test code for input errors
        if elem[4] not in data_dict:
            # new customer found
            data_dict[elem[4]]={}
            data_dict[elem[4]][elem[1]]=int(elem[2])*float(elem[3])
        else:
            # customer exists
            if elem[1] in data_dict[elem[4]]:
                # invoice of the customer exists -> adding to the sum
                data_dict[elem[4]][elem[1]]=
data_dict[elem[4]][elem[1]]+int(elem[2])*float(elem[3])
```

```
            else:
                # invoice doesn't exists for the customer -> generating new invoice info
                data_dict[elem[4]][elem[1]]=int(elem[2])*float(elem[3])
    except Exception:
         print (" Input Data Error ... -> Dictionary not generated or Inaccessible data list ")
         sys.exit(1)
#@print data_list       # debugging test code
resultsAVG={}       # dictionary for final tasks but dataset is expendable
# segregating final informatics
for customer in data_dict:
    # creating a workable dictionary for further Jobs

resultsAVG[customer]=sum(data_dict[customer].values())/len(data_dict[customer].values()
)

resultsErosiv = resultsAVG # erosive dictionary for sort and print
#helping function
def maxElem(a={}):
    #function to find out maximum valued key
    key=a.keys()[0] #raw initialization
    val=0
    for i in a:
        if a[i] > val:
            key=i
            val=a[i]
    # yielding results
    return key
    #end of function

# sort and print using erosive dictionary
steps=len(resultsErosiv)
for i in range(steps):
    t=maxElem(resultsErosiv)
    #@print t
    #@print resultsErosiv[t]
    try:
        # test removability of hashed data component
        val=resultsErosiv.pop(t)
    except Exception:
        print (" Data Error:: --> Inadequate availability  ")
        sys.exit(1)

    print('%s\t%f'%(t,val))
#
# end of job for analysis2 job5

#@ are test lines for manual debugging codes
```

```
# end of code

##
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign2.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex6.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign2ex6.py" -input
/assign1/OnlineRetail.txt -output /assign1/pythonMR_jobs/analysis2pmr/Job6
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis2pmr/Job6/part*|head -n 10
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-11-08 11:20:23,009 WARN util.NativeCodeLoader: Unable to load native-hadoop
library for your platform... using builtin-java classes where applicable
#16446   84236.250000
#12346   77183.600000
#15749   14844.766667
#15098   13305.500000
#12357   6207.670000
#12415   5948.310952
#12590   4932.130000
#12688   4873.810000
#12752   4366.780000
#18102   4327.621667
#cat: Unable to write to output stream.
#kali@kali:~$
#**..
#*** note that this job limited to 10 outputs from the head of the file systems in storage
to avoid printing 4339 records cluttering the shell screen
# ** this restriction imposed to limit output generated a cat error from the resulting
stream
```

Hive Code

```
-- Analysis 2:
-- Sales Metrics like NumCustomers, NumTransactions, AvgNumItems, MinAmtperCustomer,
MaxAmtperCustomer, AvgAmtperCustomer.. by country for top 5 countries

-- start of codes
-- one time jobs
-- **## Please avoid lines here on forward if retaildb is available in 'show databases' command and
contains data                  ##**
```

```
-- **## if avoiding these lines till line 23 directly jump to codes in analysis section of the code which
next to this section      ##**
create database RetailDB;
use RetailDB;
-- table creation
create table data_raw_headless (InvoiceNo string,StockCode string,Description string,Quantity
int,InvoiceDate string,UnitPrice float,CustomerID string,Country string) row format delimited fields
terminated by '\t' lines terminated by '\n' tblproperties("skip.header.line.count"="1");
load data inpath 'hdfs://localhost:9000/user/hive/warehouse/OnlineRetail.txt' into table
data_raw_headless;
--check number of datarows
select count (stockcode) from data_raw_headless1 where stockcode!='';
-- result 541909
--cleaning of debris
create table data_cleaned as select * from data_raw_headless1
where (quantity>0 AND unitprice>=0.0 AND customerid != '');
--confirmation and count of cleaned data
show tables;
select count (stockcode) from data_cleaned where stockcode!='';
-- result 397924
-- cleaned of debris

--Analysis Jobs
--*
-- Job1: Number of customers by country for top 5
create table kvp_asses21 as select distinct (country),(customerid) from data_cleaned;   -- kv pair
country to customer id
create table analysis21result(country string,num_customers int);    -- result master table for counter
insert into analysis21result select country,count(customerid) as counter from kvp_asses21 group by
country order by counter desc; -- counting and ordering with insertion into table
create table analysis21T5 row format delimited fields terminated by '\t' stored as textfile as select *
from analysis21result order by num_customers desc limit 5 -- final required result with segregation
and ordering
--*
--Job2: Number of transactions by Country for top 5
create table kvp_asses22 as select distinct (country),(invoiceno) from data_cleaned;   -- kv pair
country to invoice number
create table analysis22result(country string,num_transact int);    -- result master table for counter
insert into analysis22result select country,count(invoiceno) as counter from kvp_asses22 group by
country order by counter desc; -- counting and ordering with insertion into table
create table analysis22T5 row format delimited fields terminated by '\t' stored as textfile as select *
from analysis22result order by num_transact desc limit 5 -- final required result with segregation
and ordering
--*
--Job3: Average Number of items by Country for top 5
create table kvp_asses23 as select (country),(quantity) from data_cleaned;   -- kv pair country to
quantity
create table analysis23result_t(country string,num_items int);    -- result master table for counter
```

```
insert into analysis23result_t select country,sum(quantity) as total from kvp_asses23 group by
country order by total desc; -- counting and ordering with insertion into table
create table analysis23result(country string,average float); -- final table of results
insert into analysis23result select country,num_items/(select count(*) from analysis23result_t) as
average from analysis23result_T order by average desc; -- final result
create table analysis23T5 row format delimited fields terminated by '\t' stored as textfile as select *
from analysis23result order by average desc limit 5 -- final required result with segregation and
ordering
--*
--Job4,5,6: Minimum, Maximum, Average amount per customers
create table kv_asses2x456(customerid string,invoiceno string,inv_cost float); -- master table for
calucltions of job 4,5,6.
insert into kv_asses2x456 select customerid,invoiceno,sum(quantity*unitprice) from data_cleaned
group by invoiceno,customerid; -- populating table
create table analysis24result as select customerid,min(inv_cost) as minval from kv_asses2x456 group
by customerid order by minval desc; -- minimum of orders
create table analysis25result as select customerid,max(inv_cost) as maxval from kv_asses2x456 group
by customerid order by maxval desc; -- maximum of orders
create table analysis26result as select customerid,avg(inv_cost) as avgval from kv_asses2x456 group
by customerid order by avgval desc; -- average of orders
--*
-- end of code

-- results obtained
--** HIVE shell
--

--hive> select * from analysis21T5;
--OK
--United Kingdom  3921
--Germany 94
--France  87
--Spain   30
--Belgium 25
--Time taken: 0.186 seconds, Fetched: 5 row(s)
--hive> select * from analysis22T5;
--OK
--United Kingdom  16649
--Germany 457
--France  389
--EIRE    260
--Belgium 98
--Time taken: 0.153 seconds, Fetched: 5 row(s)
--hive> select * from analysis23T5;
--OK
--United Kingdom  115391.13
--Netherlands     5430.7295
--EIRE    3797.973
```

```
--Germany 3223.3242
--France  3012.7568
--Time taken: 0.18 seconds, Fetched: 5 row(s)
--hive> select * from analysis24result limit 10;
--OK
--12346   77183.59
--15749   7837.5
--12357   6207.67
--12688   4873.81
--12752   4366.78
--18251   4314.7197
--12536   4161.06
--12378   4008.6199
--15195   3861.0
--12435   3850.9
--Time taken: 0.157 seconds, Fetched: 10 row(s)
--hive> select * from analysis25result limit 10;
--OK
--16446   168469.6
--12346   77183.59
--15098   38970.0
--17450   31698.16
--12415   22775.93
--18102   22206.0
--15749   21535.9
--14646   20277.92
--12931   18841.48
--14156   16774.72
--Time taken: 0.148 seconds, Fetched: 10 row(s)
--hive> select * from analysis26result limit 10;
--OK
--16446   84236.24687504768
--12346   77183.59375
--15749   14844.7666015625
--15098   13305.5
--12357   6207.669921875
--12415   5948.310857863653
--12590   4932.1298828125
--12688   4873.81005859375
--12752   4366.77978515625
--18102   4327.621684646607
--Time taken: 0.156 seconds, Fetched: 10 row(s)
--hive>
--
--***** note that for jobs 4, 5, 6 the viewed results are limited to top 10 outputs to avoid printing all
4339 results ****
```

Pig Code

```
/* Analysis 2
Sales Metrics like NumCustomers, NumTransactions, AvgNumItems,
MinAmtperCustomer, MaxAmtperCustomer, AvgAmtperCustomer.
.. by country for top 5 countries  */

-- start of code

--raw data load with defined schema
data_raw= LOAD '/home/kali/Hadoop/Local_Datasets/OnlineRetail.txt' USING PigStorage() as
(InvoiceNo:chararray,StockCode:chararray,Description:chararray,Quantity:int,InvoiceDate:Datetime,Unit
Price:float,CustomerID:chararray,Country:chararray);
data_cleaned = FILTER data_raw BY (Quantity>=0 AND UnitPrice>=0.0 AND CustomerID!=''); --
cleaning with condition
ctrRawG= GROUP data_raw ALL;  -- grouping raw to count
ctrClnG= GROUP data_cleaned ALL; -- grouping cleaned to count
ctrRaw= FOREACH ctrRawG GENERATE COUNT(data_raw.Quantity);  --generating count raw
ctrCln= FOREACH ctrClnG GENERATE COUNT(data_cleaned.Quantity); -- generating count cleaned
-- dumping to trigger results' calculation
dump ctrRaw --value :: (541909)
dump ctrCln --value :: (397924)
-- cleaned of debris in data

--analysis job
-- Job1
-- Number of Customers by Country for top 5
kvp_asgn2x1 = FOREACH data_cleaned GENERATE Country, CustomerID; -- kv map for generating
number of customers
kvp_asgn2x1_NR = DISTINCT kvp_asgn2x1; -- non repetition ... removed all repitition of data from
kvp_asgn2x1
resGrp2x1 = GROUP  kvp_asgn2x1_NR BY Country ; -- grouping by country
resGrp2x1_final = FOREACH resGrp2x1 GENERATE group as
(Country:chararray),COUNT(kvp_asgn2x1_NR.CustomerID) as (NumCust:int); --count of number of
customers by country
ordered2x1 = ORDER resGrp2x1_final BY NumCust DESC; --ordering by descending order
order2x1L5 = LIMIT ordered2x1 5;   -- top 5 values **-- final result job1
--
-- Job2
-- Number of Transactions by Country for top 5
kvp_asgn2x2 = FOREACH data_cleaned GENERATE Country, InvoiceNo; -- kv map for generating
number of transactions
kvp_asgn2x2_NR = DISTINCT kvp_asgn2x2; -- non repetition ... removed all repitition of data from
kvp_asgn2x2
resGrp2x2 = GROUP  kvp_asgn2x2_NR BY Country ; -- grouping by country
```

```
resGrp2x2_final = FOREACH resGrp2x2 GENERATE group as
(Country:chararray),COUNT(kvp_asgn2x2_NR.InvoiceNo) as (NTransact:int); --count of number of
transactions by country
ordered2x2 = ORDER resGrp2x2_final BY NTransact DESC; --ordering by descending order
order2x2L5 = LIMIT ordered2x2 5;   -- top 5 values **-- final result job2
--
-- Job3
-- Number of Average Number of Items by Country for top 5
kvp_asgn2x3 = FOREACH data_cleaned GENERATE Country, Quantity; -- kv map for generating
number of transactions
resGrp2x3 = GROUP  kvp_asgn2x3 BY Country ; -- grouping by country
resGrp2x3_final = FOREACH resGrp2x3 GENERATE group as
(Country:chararray),SUM(kvp_asgn2x3.Quantity) as (TQuantity:int); --sum of number of items by
country
ordered2x3 = ORDER resGrp2x3_final BY TQuantity DESC; --ordering by descending order
order2x3L5x = LIMIT ordered2x3 5;   -- top 5 values total
ctrCopy = FOREACH ordered2x3 GENERATE Country; -- creating copy of ordered2x3 for counter
grouping
ctrGrp = GROUP ctrCopy ALL; --grouped
nosCtry = FOREACH ctrGrp GENERATE COUNT(ctrCopy.Country) as (value:int); -- total number of
countries
order2x3L5 = FOREACH order2x3L5x GENERATE Country as (Country:chararray),
TQuantity*1.0/(nosCtry.value) as (AverageNos:float); -- top 5 values **-- final result job3
--
--Job4 Job5 Job6
-- Minimum, Maximum, Average amount per customers
kvp_asgn2x456CI = FOREACH data_cleaned GENERATE CustomerID,InvoiceNo;--kv map for Customer
mapped to invoices ** contains duplicates
kvp_asgn2x456IE = FOREACH data_cleaned GENERATE InvoiceNo as
(InvoiceNo:chararray),UnitPrice*Quantity as (Expend:Float);--kv map for Invoices mapped to
expenditures
resGrp2x456CI = DISTINCT kvp_asgn2x456CI; -- removing duplicates from customer-> invoice indexes
resGrp2x456IE_gp = GROUP kvp_asgn2x456IE BY InvoiceNo; -- pre grouping for calculating the sum
total of an invoice bill
resGrp2x456IE = FOREACH resGrp2x456IE_gp GENERATE group as
(InvoiceNo:chararray),SUM(kvp_asgn2x456IE.Expend) as (TotalAtInv:float); -- summing to total of every
invoices
masterJoin2x456 = JOIN resGrp2x456IE BY InvoiceNo, resGrp2x456CI BY InvoiceNo; -- joining by
common key -> viz, Invoice number
masterCollect2x456 = FOREACH masterJoin2x456 GENERATE resGrp2x456CI::CustomerID as
(CustomerID:chararray),resGrp2x456IE::TotalAtInv as (TotalCost:float); -- link customers to total costs at
a invoice collection
masterGrp2x456 = GROUP masterCollect2x456 BY CustomerID; -- grouped to get the specific tasks
done
minCosts = FOREACH masterGrp2x456 GENERATE group as
(CustomerID:chararray),MIN(masterCollect2x456.TotalCost) as (Minima:Float); -- job4 precursor
maxCosts = FOREACH masterGrp2x456 GENERATE group as
(CustomerID:chararray),MAX(masterCollect2x456.TotalCost) as (Maxima:Float); -- job5 precursor
```

```
avgCosts = FOREACH masterGrp2x456 GENERATE group as
(CustomerID:chararray),SUM(masterCollect2x456.TotalCost)/SIZE(masterCollect2x456.TotalCost) as
(Average:Float); -- job6 precursor
order2x4 = ORDER minCosts BY Minima DESC;  -- descrnding order ** -- final result job4
order2x5 = ORDER maxCosts BY Maxima DESC;  -- descrnding order ** -- final result job5
order2x6 = ORDER avgCosts BY Average DESC;  -- descrnding order ** -- final result job6
--
-- Storing the final results in respective locations pertaining to job
STORE order2x1L5 INTO '/home/kali/Hadoop/Results/pig_results/analysis2/Job1' USING PigStorage();
STORE order2x2L5 INTO '/home/kali/Hadoop/Results/pig_results/analysis2/Job2' USING PigStorage();
STORE order2x3L5 INTO '/home/kali/Hadoop/Results/pig_results/analysis2/Job3' USING PigStorage();
STORE order2x4 INTO '/home/kali/Hadoop/Results/pig_results/analysis2/Job4' USING PigStorage();
STORE order2x5 INTO '/home/kali/Hadoop/Results/pig_results/analysis2/Job5' USING PigStorage();
STORE order2x6 INTO '/home/kali/Hadoop/Results/pig_results/analysis2/Job6' USING PigStorage();

-- end of code

/*
** Results Obtained
#**
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis2/Job1/part*
United Kingdom  3921
Germany 94
France  87
Spain   30
Belgium 25
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis2/Job2/part*
United Kingdom  16649
Germany 457
France  389
EIRE    260
Belgium 98
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis2/Job3/part*
United Kingdom  115391.13
Netherlands     5430.7295
EIRE    3797.973
Germany 3223.3242
France  3012.7568
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis2/Job4/part*|head -n 10
12346   77183.59
15749   7837.5
12357   6207.67
12688   4873.81
12752   4366.78
18251   4314.7197
12536   4161.06
12378   4008.62
15195   3861.0
```

```
12435   3850.9
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis2/Job5/part*|head -n 10
16446   168469.6
12346   77183.59
15098   38970.0
17450   31698.16
12415   22775.93
18102   22206.0
15749   21535.9
14646   20277.92
12931   18841.48
14156   16774.72
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis2/Job6/part*|head -n 10
16446   84236.25
12346   77183.59
15749   14844.767
15098   13305.5
12357   6207.67
12415   5948.311
12590   4932.13
12688   4873.81
12752   4366.78
18102   4327.6216
kali@kali:~$

#**

***** note that for jobs 4, 5, 6 the viewed results are limited to top 10 outputs to avoid printing all
4339 results ****
*/
```

Spark Code

```
//Spark code for Analysis2:
// Sales Metrics like NumCustomers, NumTransactions, AvgNumItems, MinAmtperCustomer,
MaxAmtperCustomer, AvgAmtperCustomer. .. by country for top 5 countries

//cleaning of visible debris
var data_raw= sc.textFile("hdfs://localhost:9000/assign1/OnlineRetail.txt")
var data_raw_split= data_raw.map(x=>x.split("\t"))
var data_headless=data_raw_split.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1)
else iter }
data_headless.count
var data_cleaned=data_headless.filter{x=> if((x(3).toInt >=0 ) && (x(5).toFloat >=0.0) &&
(x(6)!=""))true else false}
data_cleaned.count
// cleaned data of primary debris
```

```
//analysis jobs

//process 1: number of customers by countries for top 5
var cust2ctry_dup=data_cleaned.map(x=> {(x(7), x(6), 1)}) //customer id mapped to country::
contains duplicates
var cust2ctry=cust2ctry_dup.distinct  //removed duplicates
var kvp_cust2ctry=cust2ctry.map(x=> {(x._1, x._3)})  // map out occurences
var num_cust=kvp_cust2ctry.reduceByKey((a,b)=>(a+b))  // country to number of customers ::
frequency
var num_cust_sorted=num_cust.sortBy(_._2,false)    // sort
var soln1=num_cust_sorted.take(5) // result1

//process 2: number of transactions by countries for top 5
var inv2ctry_dup=data_cleaned.map(x=> {(x(7), x(0), 1)}) // invoice mapped to country ::  contains
duplicates
var inv2ctry=inv2ctry_dup.distinct //removed duplicates
var kvp_inv2ctry=inv2ctry.map(x=> {(x._1, x._3)}) //map out occurences
var num_tran=kvp_inv2ctry.reduceByKey((a,b)=>(a+b)) // country to number of transactions ::
frequency
var num_tran_sorted=num_tran.sortBy(_._2,false)  //sort
var soln2=num_tran_sorted.take(5) //result2

//process 3: average number of items by country for top 5
var kvp_qty2ctry=data_cleaned.map(x=> {(x(7), x(3).toInt)}) //quantity mapped to country
var kvp_tQ2ctry=kvp_qty2ctry.reduceByKey(_+_) // total quantity to each country
var nosCtry=kvp_tQ2ctry.count // total number of countries
var tQ_sorted=kvp_tQ2ctry.sortBy(_._2,false)  // sort
var tQ_t5=tQ_sorted.take(5) // top 5 totals
var soln3_vec= {for(i <- 0 to 4) yield (tQ_t5(i)._1,tQ_t5(i)._2*1.0 /nosCtry)}  // calculating for the top 5
totals
var soln3=soln3_vec.toArray  //result3

//process 4,5,6: Minimum, Maximum, Average amount per customers
var inv2cost_nr = data_cleaned.map(x=>{(x(0),x(3). toInt * x(5).toFloat)})  // invoices mapped to the
cost -- non reduced
var inv2cust_dup=data_cleaned.map(x=>{(x(0),x(6))})  // invoices mapped to customers -- contains
duplicates
var inv2cust=inv2cust_dup.distinct // removing duplicates
var inv2cost=inv2cost_nr.reduceByKey(_+_) // summing up costs by invoices
var mainCollect_join=inv2cust.join(inv2cost) // purposefully made invoice as the key since join works
on K,V data inputs
var cust2cost=mainCollect_join.map(x=> x._2) // customer mapped to costs incurred
var cust2cost_grp=cust2cost.groupByKey() // group to create iterable set of costs
var minCosts=cust2cost_grp.map(x=> (x._1,x._2.min))  // minimum costs per customers
var maxCosts=cust2cost_grp.map(x=> (x._1,x._2.max))  // maximum costs per customers
var avgCosts=cust2cost_grp.map(x=> (x._1,x._2.sum/x._2.size))  // average costs per customers
var soln4=minCosts.sortBy(_._2,false) //sort descending   // result 4
var soln5=maxCosts.sortBy(_._2,false) //sort descending   // result 5
var soln6=avgCosts.sortBy(_._2,false) //sort descending   // result 6

// save operations
sc.parallelize(soln1).saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis2/Job1")   //
number of customers
sc.parallelize(soln2).saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis2/Job2")   //
number of transactions
sc.parallelize(soln3).saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis2/Job3")   //
average number of items
soln4.saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis2/Job4")   // minimum
payout per invoice
soln5.saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis2/Job5")   // maximum
payout per invoice
soln6.saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis2/Job6")   // average payout
per invoice

//end of code


// Solutions Obtained
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis2/Job1/part*
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-06 13:45:31,716 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
//(United Kingdom,3921)
//(Germany,94)
//(France,87)
//(Spain,30)
//(Belgium,25)
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis2/Job2/part*
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-06 13:45:50,012 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
//(United Kingdom,16649)
//(Germany,457)
//(France,389)
//(EIRE,260)
//(Belgium,98)
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis2/Job3/part*
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-06 13:46:09,528 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
//(United Kingdom,115391.13513513513)
//(Netherlands,5430.72972972973)
//(EIRE,3797.972972972973)
//(Germany,3223.324324324324)
//(France,3012.7567567567567)
```

```
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis2/Job4/part*|head -n 10
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-06 13:46:34,308 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
//(12346,77183.59)
//(15749,7837.5)
//(12357,6207.6714)
//(12688,4873.8096)
//(12752,4366.78)
//(18251,4314.7197)
//(12536,4161.06)
//(12378,4008.6196)
//(15195,3861.0)
//(12435,3850.9)
//cat: Unable to write to output stream.
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis2/Job5/part*|head -n 10
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-06 13:47:02,137 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
//(16446,168469.6)
//(12346,77183.59)
//(15098,38970.0)
//(17450,31698.16)
//(12415,22775.93)
//(18102,22206.0)
//(15749,21535.9)
//(14646,20277.92)
//(12931,18841.48)
//(14156,16774.72)
//cat: Unable to write to output stream.
//cat: Unable to write to output stream.
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis2/Job6/part*|head -n 10
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-06 13:47:20,295 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
//(16446,84236.25)
//(12346,77183.59)
//(15749,14844.767)
//(15098,13305.5)
//(12357,6207.6714)
//(12415,5948.3105)
//(12590,4932.1304)
//(12688,4873.8096)
//(12752,4366.78)
//(18102,4327.621)
//cat: Unable to write to output stream.
//cat: Unable to write to output stream.
//kali@kali:~$
```

```
///
//*** note that jobs 4, 5, 6, are limited to 10 outputs from the head of the file systems in storage to
avoid printing 4339 records cluttering the shell screen
// ** this restriction imposed to limit output generated a cat error from the resulting stream
```

Question 3.

Python Mapper

```
#!usr/bin/python
#
# *******
# **    Analysis 3:                                                          *
# *   Daily sales activity like numvisits, total amount monthly and quarterly for 1 year.**
# *******
# start of code
import sys
for line in sys.stdin:
   row=line.strip()
   components=row.split("\t")
   try:
      if ( "" not in components and int(components[3])>= 0 and float(components[5])>= 0.0 ):

         # generating K,V pairs after filtering out debris in data

print('%s\t%s\t%d\t%f'%(components[4],components[0],int(components[3]),float(components[5])))
      else:
        #print(components)
        continue
   except Exception:
      #print(components)
      continue


# end of code
#results mentioned in reducer
```

Python Reducers

1.
```
#!usr/bin/python
#
# *******
# **    Analysis 3:
```

```
# *   Daily sales activity like numvisits, total amount monthly and quarterly for 1 year.**
# *******
# **** Specific operation:  Number Visits Monthly ****
# start of code
import sys
from operator import itemgetter
data_list=[]
for line in sys.stdin:
    data_line=line.strip()
    elements=data_line.split("\t")
    data_list.append(elements)  # generate local list
if len(data_list) == 0:
    print(" Input Stream Error ... -> Reducer didn't receive any data ")
    sys.exit(1)
data_collect=[]  # storing dictionary for all primary k-v sets
for elem in data_list:
    try:
        # test code for input errors
        if (elem[0],elem[1]) not in data_collect:
            # new found unique key :: country
            data_collect.append((elem[0],elem[1]))
        # intake of unique key value sets

    except Exception:
         print (" Input Data Error ... -> Dictionary not generated or Inaccessible data list ")
         sys.exit(1)

soln31={}          # final kv pairing and collecting results
for elem in data_collect:
    key=int(elem[0][5:7])
    if key in soln31:
        soln31[key].append(elem[1])
    else:
        soln31[key]=[elem[1]]

# generating and printing results
for elem in sorted(soln31):
    print('%s\t%d'%(elem,len(soln31[elem])))



# end of code

##
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
```

```
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign3.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign3.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign3ex1.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign3ex1.py" -input /assign1/OnlineRetail.txt -
output /assign1/pythonMR_jobs/analysis3pmr/Job1
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis3pmr/Job1/part*
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#020-11-19 19:50:21,686 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
#1      993
#2      1003
#3      1324
#4      1153
#5      1559
#6      1394
#7      1331
#8      1283
#9      1757
#10     1930
#11     2660
#12     2179
#kali@kali:~$
#
```

2.

```
#!usr/bin/python
#
# *******
# **    Analysis 3:
# *   Daily sales activity like numvisits, total amount monthly and quarterly for 1 year.**
# *******
# **** Specific operation:  Number Visits Quarterly ****
# start of code
import sys
from operator import itemgetter
data_list=[]
for line in sys.stdin:
    data_line=line.strip()
    elements=data_line.split("\t")
    data_list.append(elements)  # generate local list
if len(data_list) == 0:
    print(" Input Stream Error ... -> Reducer didn't receive any data ")
    sys.exit(1)
data_collect=[]  # storing dictionary for all primary k-v sets
for elem in data_list:
    try:
        # test code for input errors
        if (elem[0],elem[1]) not in data_collect:
            # new found unique key :: country
            data_collect.append((elem[0],elem[1]))
        # intake of unique key value sets

    except Exception:
         print (" Input Data Error ... -> Dictionary not generated or Inaccessible data list ")
         sys.exit(1)

soln32={}          # final kv pairing and collecting results
for elem in data_collect:
    key=int((float(elem[0][5:7])-0.1)/3)
    if key in soln32:
        soln32[key].append(elem[1])
    else:
        soln32[key]=[elem[1]]

# generating and printing results
for elem in sorted(soln32):
    print('%s\t%d'%(elem,len(soln32[elem])))



# end of code

##
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign3.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign3.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign3ex2.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign3ex2.py" -input /assign1/OnlineRetail.txt -
output /assign1/pythonMR_jobs/analysis3pmr/Job2
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis3pmr/Job2/part*
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-11-19 20:47:13,092 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
#0      3320
#1      4106
#2      4371
#3      6769
#kali@kali:~$
```

```
##
```

3.
```
#!usr/bin/python
#
# *******
# **    Analysis 3:
# *   Daily sales activity like numvisits, total amount monthly and quarterly for 1 year.**
# *******
# **** Specific operation:  TotalCosts Monthly ****
# start of code
import sys
from operator import itemgetter
data_list=[]
for line in sys.stdin:
   data_line=line.strip()
   elements=data_line.split("\t")
   data_list.append(elements)  # generate local list
if len(data_list) == 0:
   print(" Input Stream Error ... -> Reducer didn't receive any data ")
   sys.exit(1)
data_dict={}  # storing all costs
for elem in data_list:
   try:
      # test code for input errors
      if elem[0] not in data_dict:
         # new found unique key :: date
         data_dict[elem[0]]=int(elem[2])*float(elem[3])
      else:
         #  already available date key
         data_dict[elem[0]] = data_dict[elem[0]]+int(elem[2])*float(elem[3])

   except Exception:
       print (" Input Data Error ... -> Dictionary not generated or In-accessable data list ")
       sys.exit(1)

soln33={}          # final kv pairing and collecting results
for elem in data_dict:
   key=int(elem[5:7])
   if key in soln33:
      soln33[key]=soln33[key]+data_dict[elem]
   else:
      soln33[key]=data_dict[elem]

# generating and printing results
for elem in sorted(soln33):
   print('%s\t%f'%(elem,soln33[elem]))
```

```
# end of code

##
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign3.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign3.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign3ex3.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign3ex3.py" -input /assign1/OnlineRetail.txt -
output /assign1/pythonMR_jobs/analysis3pmr/Job3
# results obtained
##**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis3pmr/Job3/part*
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-11-19 20:58:21,152 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
#1      569445.040000
#2      447137.350000
#3      595500.760000
#4      469200.361000
#5      678594.560000
#6      661213.690000
#7      600091.011000
#8      645343.900000
#9      952838.382000
#10     1039318.790000
#11     1161817.380000
#12     1090906.680000
#kali@kali:~$

#
#
```

4.
```
#!usr/bin/python
#
# *******
# **    Analysis 3:
# *   Daily sales activity like numvisits, total amount monthly and quarterly for 1 year.**
# *******
# **** Specific operation:  TotalCosts Quarterly ****
# start of code
import sys
from operator import itemgetter
```

```
data_list=[]
for line in sys.stdin:
   data_line=line.strip()
   elements=data_line.split("\t")
   data_list.append(elements)  # generate local list
if len(data_list) == 0:
   print(" Input Stream Error ... -> Reducer didn't receive any data ")
   sys.exit(1)
data_dict={}  # storing all costs
for elem in data_list:
   try:
      # test code for input errors
      if elem[0] not in data_dict:
         # new found unique key :: date
         data_dict[elem[0]]=int(elem[2])*float(elem[3])
      else:
         #  already available date key
         data_dict[elem[0]] = data_dict[elem[0]]+int(elem[2])*float(elem[3])

   except Exception:
        print (" Input Data Error ... -> Dictionary not generated or In-accessable data list ")
        sys.exit(1)

soln34={}          # final kv pairing and collecting results
for elem in data_dict:
   key=int((float(elem[5:7])-0.1)/3)
   if key in soln34:
      soln34[key]=soln34[key]+data_dict[elem]
   else:
      soln34[key]=data_dict[elem]

# generating and printing results
for elem in sorted(soln34):
   print('%s\t%f'%(elem,soln34[elem]))



# end of code

##
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign3.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign3.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign3ex4.py -reducer "python
```

```
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign3ex4.py" -input /assign1/OnlineRetail.txt -
output /assign1/pythonMR_jobs/analysis3pmr/Job4
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis3pmr/Job4/part*
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-11-19 21:03:39,151 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
#0      1612083.150000
#1      1809008.611000
#2      2198273.293000
#3      3292042.850000
#kali@kali:~$
#
```

Hive Codes

```
-- Analysis 3:
-- ** Daily sales activity like numvisits, totalamount monthly and quarterly for 1 year ***

-- start of codes
-- one time jobs
-- **## Please avoid lines here on forward if retaildb is available in 'show databases' command and
contains data                  ##**
-- **## if avoiding these lines till line 23 directly jump to codes in analysis section of the code which next
to this section      ##**
create database RetailDB;
use RetailDB;
-- table creation
create table data_raw_headless (InvoiceNo string,StockCode string,Description string,Quantity
int,InvoiceDate string,UnitPrice float,CustomerID string,Country string) row format delimited fields
terminated by '\t' lines terminated by '\n' tblproperties("skip.header.line.count"="1");
load data inpath 'hdfs://localhost:9000/user/hive/warehouse/OnlineRetail.txt' into table
data_raw_headless;
--check number of datarows
select count (stockcode) from data_raw_headless1 where stockcode!='';
-- result 541909
--cleaning of debris
create table data_cleaned as select * from data_raw_headless1
where (quantity>0 AND unitprice>=0.0 AND customerid != '');
--confirmation and count of cleaned data
show tables;
select count (stockcode) from data_cleaned where stockcode!='';
-- result 397924
-- cleaned of debris
```

```
--Analysis Jobs
--*
-- Job1: NumVisits monthly
create table kv_asses31(timeval bigint, invoiceno string); -- creating schema kv pair timestamp to invoice
insert into kv_asses31 select distinct (unix_timestamp(invoicedate,'yyyy-MM-dd HH:mm')),invoiceno from
data_cleaned; -- populating table
create table monthlyhits(month int,invoice string); -- segregation table schema
insert into monthlyhits select month(from_unixtime(timeval)), invoiceno from kv_asses31;
create table soln31 as select month,count(invoice) from monthlyhits group by month order by month; --
final result ;

-- Job2: NumVisits quarterly
create table quarterlyhits (quarter int,invoice string); --segregation table schema
insert into quarterlyhits select floor((month(from_unixtime(timeval))-0.1)/3), invoiceno from kv_asses31; --
filling up
create table soln32 as select quarter,count(invoice) from quarterlyhits group by quarter order by quarter; -
- final result ;

--Job3: TotalCosts monthly
create table kv_asses32(timeval bigint, totalcost float); -- creating schema kv pair timestamp to cost
insert into kv_asses32 select (unix_timestamp(invoicedate,'yyyy-MM-dd HH:mm')),(quantity*unitprice)
from data_cleaned; -- populating table
create table monthlycosts(month int,costs float); -- segregation table schema
insert into monthlycosts select month(from_unixtime(timeval)), totalcost from kv_asses32; -- filling
create table soln33 as select month,sum(costs) from monthlycosts group by month order by month; --
final result;

-- Job4: TotalCosts quarterly
create table quarterlycosts (quarter int,costs float); --segregation table schema
insert into quarterlycosts select floor((month(from_unixtime(timeval))-0.1)/3), totalcost from kv_asses32; --
filling up
create table soln34 as select quarter,sum(costs) from quarterlycosts group by quarter order by quarter; --
final result ;
--*;

-- end of code

-- results obtained
--** HIVE shell
--

-- Number of visits per month
--hive> select * from soln31;
--OK
--1      993
--2      1003
--3      1324
--4      1153
```

```
--5      1559
--6      1394
--7      1331
--8      1283
--9      1757
--10      1930
--11      2660
--12      2179
--Time taken: 0.155 seconds, Fetched: 12 row(s)
--hive>

-- Number of visits per quarter
--hive> select * from soln32;
--OK
--0      3320
--1      4106
--2      4371
--3      6769
--Time taken: 0.161 seconds, Fetched: 4 row(s)
--hive>

-- Total Costs  Monthly
--hive> select * from soln33;
--OK
--1      569445.0322882384
--2      447137.3490101546
--3      595500.7586191893
--4      469200.3588609899
--5      678594.5577653646
--6      661213.6881134436
--7      600091.0089869479
--8      645343.8979516104
--9      952838.3793331122
--10      1039318.7866888493
--11      1161817.3760578
--12      1090906.6718007103
--Time taken: 0.169 seconds, Fetched: 12 row(s)
--hive>


-- Total Costs Quarterly
--hive> select * from soln34;
--OK
--0      1612083.1399175823
--1      1809008.6047397982
--2      2198273.286271671
--3      3292042.8345473595
--Time taken: 0.16 seconds, Fetched: 4 row(s)
```

```
--hive>


--********
```

Pig Codes

```
/* Analysis 3:
Daily sales activity like numvisits, totalamount monthly and quarterly for 1 year. */

-- start of code

--raw data load with defined schema
data_raw= LOAD '/home/kali/Hadoop/Local_Datasets/OnlineRetail.txt' USING PigStorage() as
(InvoiceNo:chararray,StockCode:chararray,Description:chararray,Quantity:int,InvoiceDate:Datetime,Unit
Price:float,CustomerID:chararray,Country:chararray);
data_cleaned = FILTER data_raw BY (Quantity>=0 AND UnitPrice>=0.0 AND CustomerID!=''); --
cleaning with condition
ctrRawG= GROUP data_raw ALL;  -- grouping raw to count
ctrClnG= GROUP data_cleaned ALL; -- grouping cleaned to count
ctrRaw= FOREACH ctrRawG GENERATE COUNT(data_raw.Quantity);  --generating count raw
ctrCln= FOREACH ctrClnG GENERATE COUNT(data_cleaned.Quantity); -- generating count cleaned
-- dumping to trigger results' calculation
dump ctrRaw --value :: (541909)
dump ctrCln --value :: (397924)
-- cleaned of debris in data

--analysis job
-- Job1 : NumVisits Monthly
kv_asses31 = FOREACH data_cleaned GENERATE InvoiceDate,InvoiceNo; -- kv pair datetime to invoice
kv_asses31NR = DISTINCT kv_asses31; -- removing duplicates
soln31_prx = FOREACH kv_asses31NR GENERATE GetMonth(InvoiceDate) as (Month:int),InvoiceNo; --
presolution
soln31_grp = GROUP soln31_prx BY Month;  -- group
soln31 = FOREACH soln31_grp GENERATE group as (Month:int),COUNT(soln31_prx.InvoiceNo);  --
final solution

-- Job2 :  NumVisits Quarterly
soln32_prx = FOREACH kv_asses31NR GENERATE FLOOR((GetMonth(InvoiceDate)-0.1)/3) as
(Quarter:int),InvoiceNo; --presolution
soln32_grp = GROUP soln32_prx BY Quarter;  -- group
soln32 = FOREACH soln32_grp GENERATE group as (Quarter:int),COUNT(soln32_prx.InvoiceNo);  --
final solution

-- Job3 : TotalCost Monthly
kv_asses32 = FOREACH data_cleaned GENERATE InvoiceDate,(Quantity*UnitPrice) as (TotalCost:float);
-- kv pair datetime to totalcost
```

```
soln33_prx = FOREACH kv_asses32 GENERATE GetMonth(InvoiceDate) as (Month:int),TotalCost; --
presolution
soln33_grp = GROUP soln33_prx BY Month;  -- group
soln33 = FOREACH soln33_grp GENERATE group as (Month:int),SUM(soln33_prx.TotalCost);  -- final
solution

-- Job4 : TotalCost Quarterly
soln34_prx = FOREACH kv_asses32 GENERATE FLOOR((GetMonth(InvoiceDate)-0.1)/3) as
(Quarter:int),TotalCost; --presolution
soln34_grp = GROUP soln34_prx BY Quarter;  -- group
soln34 = FOREACH soln34_grp GENERATE group as (Quarter:int),SUM(soln34_prx.TotalCost);  -- final
solution

--
--Storage
STORE soln31 INTO '/home/kali/Hadoop/Results/pig_results/analysis3/Job1' USING PigStorage();
STORE soln32 INTO '/home/kali/Hadoop/Results/pig_results/analysis3/Job2' USING PigStorage();
STORE soln33 INTO '/home/kali/Hadoop/Results/pig_results/analysis3/Job3' USING PigStorage();
STORE soln34 INTO '/home/kali/Hadoop/Results/pig_results/analysis3/Job4' USING PigStorage();

-- end of code

/*
** Results Obtained
#**
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis3/Job1/part*
1	993
2	1003
3	1324
4	1153
5	1559
6	1394
7	1331
8	1283
9	1757
10	1930
11	2660
12	2179
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis3/Job2/part*
0	3320
1	4106
2	4371
3	6769
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis3/Job3/part*
1	569445.0322882384
2	447137.3490101546
3	595500.7586191893
4	469200.3588609899
```

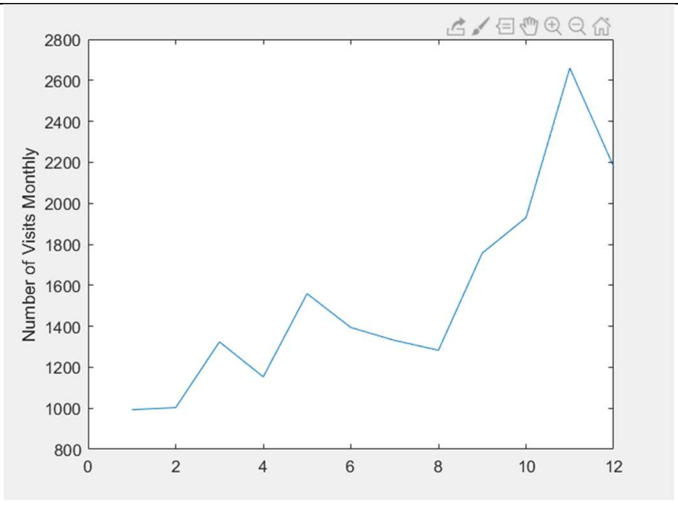
```
5       678594.5577653646
6       661213.6881134436
7       600091.0089869479
8       645343.8979516104
9       952838.3793331122
10       1039318.7866888493
11       1161817.3760578
12       1090906.6718007103
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis3/Job4/part*
0       1612083.1399175823
1       1809008.6047397982
2       2198273.286271671
3       3292042.8345473595
kali@kali:~$
#**
*/
```

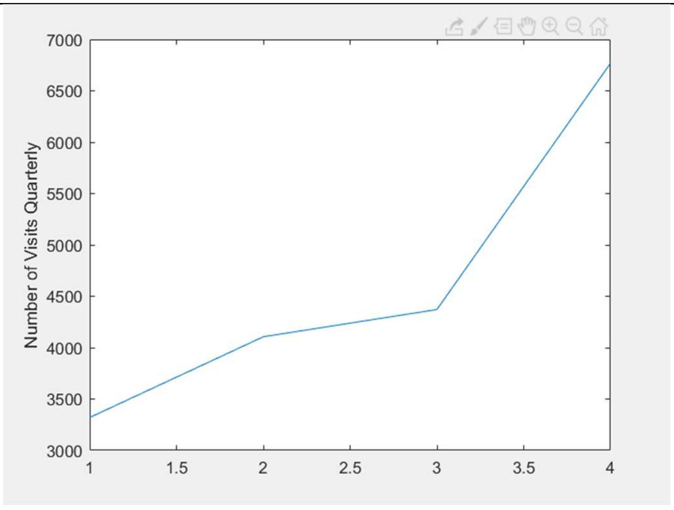
Spark Codes

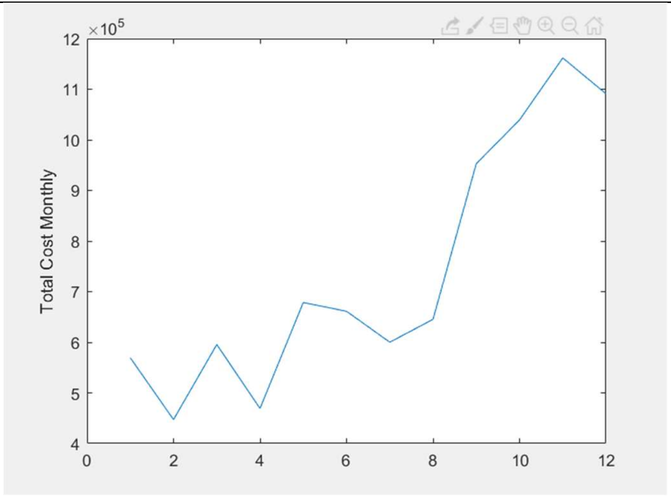
```
//Spark code for Analysis3
// **Daily sales activity like numvisits, totalamount monthly and quarterly for 1 year.
//cleaning of visible debris
var data_raw= sc.textFile("hdfs://localhost:9000/assign1/OnlineRetail.txt")
var data_raw_split= data_raw.map(x=>x.split("\t"))
var data_headless=data_raw_split.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1) else
iter }
data_headless.count
var data_cleaned=data_headless.filter{x=> if((x(3).toInt >=0 ) && (x(5).toFloat >=0.0) && (x(6)!=""))true
else false}
data_cleaned.count
// cleaned data of primary debris

//Analysis Job:
//Job1: Numvisits Monthly
var kvp_dt2inv=data_cleaned.map(x=>{(x(4),x(0))}).distinct
var kv_grp31=kvp_dt2inv.map(x=>{(x._1.substring(5,7).toInt,x._2)}).groupByKey()
var soln31=kv_grp31.map(x=>{(x._1,x._2.size)}).sortBy(_._1)

//Job2: Numvisits quarterly
var kv_grp32=kvp_dt2inv.map(x=>{(((x._1.substring(5,7).toFloat-0.1)/3).toInt,x._2)}).groupByKey()
var soln32=kv_grp32.map(x=>{(x._1,x._2.size)}).sortBy(_._1)

//Job3: TotalCosts Monthly
var kvp_dt2cost=data_cleaned.map(x=>{(x(4),(x(3).toInt*x(5).toFloat))})
var kv_grp33=kvp_dt2cost.map(x=>{(x._1.substring(5,7).toInt,x._2)}).groupByKey()
var soln33=kv_grp33.map(x=>{(x._1,x._2.sum)}).sortBy(_._1)
```

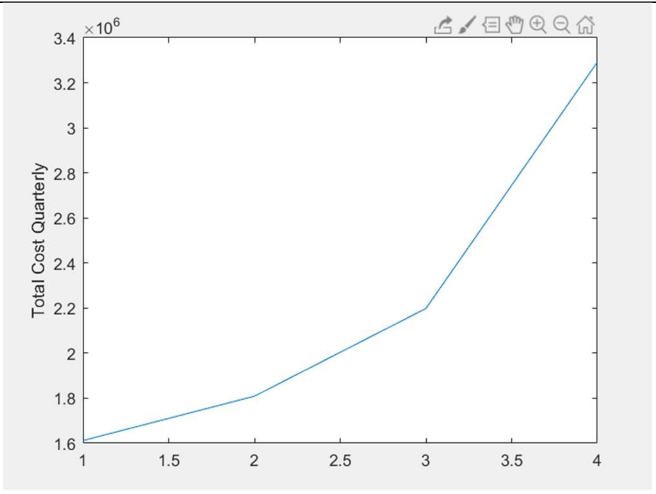
```
//Job4: TotalCosts Quarterly
var kv_grp34=kvp_dt2cost.map(x=>{(((x._1.substring(5,7).toFloat-0.1)/3).toInt,x._2)}).groupByKey()
var soln34=kv_grp34.map(x=>{(x._1,x._2.sum)}).sortBy(_._1)

//Save operations
soln31.saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis3/Job1")
soln32.saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis3/Job2")
soln33.saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis3/Job3")
soln34.saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis3/Job4")

//end of code



// Solutions Obtained
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis3/Job1/part*
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-18 21:10:41,958 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
//(1,993)
//(2,1003)
//(3,1324)
//(4,1153)
//(5,1559)
//(6,1394)
//(7,1331)
//(8,1283)
//(9,1757)
//(10,1930)
//(11,2660)
//(12,2179)
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis3/Job2/part*
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-18 21:10:55,432 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
//(0,3320)
//(1,4106)
//(2,4371)
//(3,6769)
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis3/Job3/part*
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-18 21:11:03,089 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
//(1,569433.8)
//(2,447132.38)
//(3,595491.7)
//(4,469200.16)
//(5,678592.2)
```

```
//(6,661210.6)
//(7,600090.5)
//(8,645341.4)
//(9,952817.5)
//(10,1039277.94)
//(11,1161771.2)
//(12,1090883.8)
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis3/Job4/part*
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-18 21:11:11,186 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
//(0,1612100.4)
//(1,1809007.2)
//(2,2198295.5)
//(3,3292369.5)
//kali@kali:~$
//
```









Question 4.

Python Mappers

```
#!usr/bin/python
#
# *******
# **    Analysis 4:                                           *
# *   Hourly sales activity like numvisits, totalamount per hour of day. **
# *******
# start of code
import sys
for line in sys.stdin:
    row=line.strip()
    components=row.split("\t")
    try:
        if ( "" not in components and int(components[3])>= 0 and float(components[5])>= 0.0 ):

            # generating K,V pairs after filtering out debris in data
            print('%s\t%s\t%d\t%f'%(components[4],components[0],int(components[3]),float(components[5])))
        else:
           #print(components)
           continue
    except Exception:
        #print(components)
        continue


# end of code
# results mentioned in the reducer
```

Python Reducers

1.
```
#!usr/bin/python
#
# *******
# **    Analysis 4:
# *   Hourly sales activity like numvisits, totalamount per hour of day.**
# *******
# **** Specific operation:  Number Visits Hourly ****
# start of code
import sys
from operator import itemgetter
data_list=[]
```

```
for line in sys.stdin:
    data_line=line.strip()
    elements=data_line.split("\t")
    data_list.append(elements)  # generate local list
if len(data_list) == 0:
    print(" Input Stream Error ... -> Reducer didn't receive any data ")
    sys.exit(1)
data_collect=[]  # storing dictionary for all primary k-v sets
for elem in data_list:
    try:
        # test code for input errors
        if (elem[0],elem[1]) not in data_collect:
            # new found unique key :: country
            data_collect.append((elem[0],elem[1]))
        # intake of unique key value sets

    except Exception:
         print (" Input Data Error ... -> Dictionary not generated or Inaccessible data list ")
         sys.exit(1)

soln41={}          # final kv pairing and collecting results
for elem in data_collect:
    key=elem[0][0:13]
    if key.endswith(':'): key=elem[0][0:12]

    if key in soln41:
      soln41[key].append(elem[1])
    else:
      soln41[key]=[elem[1]]

# generating and printing results
for elem in sorted(soln41):
    print('%s\t%d'%(elem,len(soln41[elem])))

#print data_collect



# end of code

##
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign4.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign4.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign4ex1.py -reducer "python
```

```
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign4ex1.py" -input /assign1/OnlineRetail.txt -
output /assign1/pythonMR_jobs/analysis4pmr/Job1
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis4pmr/Job1/part* |head -n 10
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-11-19 21:09:42,483 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
#2010-12-01 10   11
#2010-12-01 11   12
#2010-12-01 12   22
#2010-12-01 13   12
#2010-12-01 14   8
#2010-12-01 15   14
#2010-12-01 16   17
#2010-12-01 17   4
#2010-12-01 8    6
#2010-12-01 9    16
#kali@kali:~$

#
```

2.

```
#!usr/bin/python
#
# *******
# **    Analysis 4:
# *   Hourly sales activity like numvisits, totalamount per hour of day**
# *******
# **** Specific operation:  TotalCosts Hourly ****
# start of code
import sys
from operator import itemgetter
data_list=[]
for line in sys.stdin:
    data_line=line.strip()
    elements=data_line.split("\t")
    data_list.append(elements)  # generate local list
if len(data_list) == 0:
    print(" Input Stream Error ... -> Reducer didn't receive any data ")
    sys.exit(1)
data_dict={}  # storing all costs
for elem in data_list:
    try:
        # test code for input errors
        if elem[0] not in data_dict:
            # new found unique key :: date
            data_dict[elem[0]]=int(elem[2])*float(elem[3])
        else:
            #  already available date key
            data_dict[elem[0]] = data_dict[elem[0]]+int(elem[2])*float(elem[3])

    except Exception:
         print (" Input Data Error ... -> Dictionary not generated or In-accessable data list ")
         sys.exit(1)

soln42={}         # final kv pairing and collecting results
for elem in data_dict:
    key=elem[0:13]
    if key.endswith(':'): key=elem[0:12]

    if key in soln42:
        soln42[key]=soln42[key]+data_dict[elem]
    else:
        soln42[key]=data_dict[elem]

# generating and printing results
for elem in sorted(soln42):
    print('%s\t%f'%(elem,soln42[elem]))



# end of code

##
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign4.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign4.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign4ex2.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign4ex2.py" -input /assign1/OnlineRetail.txt -
output /assign1/pythonMR_jobs/analysis4pmr/Job2
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis4pmr/Job2/part* |head -n 10
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-11-19 21:13:25,308 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
#2010-12-01 10   5235.810000
#2010-12-01 11   4234.160000
#2010-12-01 12   7447.920000
#2010-12-01 13   5063.540000
#2010-12-01 14   2831.220000
#2010-12-01 15   3587.310000
```

```
#2010-12-01 16   8623.140000
#2010-12-01 17   613.190000
#2010-12-01 8    1383.810000
#2010-12-01 9    7356.390000
#cat: Unable to write to output stream.
#kali@kali:~$
#
## *** restricting output to 10 outputs results in the cat error
```

Hive Codes

```
-- Analysis 4:
-- ** Hourly sales activity like numvisits, totalamount per hour of day. ***

-- start of codes
-- one time jobs
-- **## Please avoid lines here on forward if retaildb is available in 'show databases' command and
contains data                  ##**
-- **## if avoiding these lines till line 23 directly jump to codes in analysis section of the code which next
to this section      ##**
create database RetailDB;
use RetailDB;
-- table creation
create table data_raw_headless (InvoiceNo string,StockCode string,Description string,Quantity
int,InvoiceDate string,UnitPrice float,CustomerID string,Country string) row format delimited fields
terminated by '\t' lines terminated by '\n' tblproperties("skip.header.line.count"="1");
load data inpath 'hdfs://localhost:9000/user/hive/warehouse/OnlineRetail.txt' into table
data_raw_headless;
--check number of datarows
select count (stockcode) from data_raw_headless1 where stockcode!='';
-- result 541909
--cleaning of debris
create table data_cleaned as select * from data_raw_headless1
where (quantity>0 AND unitprice>=0.0 AND customerid != '');
--confirmation and count of cleaned data
show tables;
select count (stockcode) from data_cleaned where stockcode!='';
-- result 397924
-- cleaned of debris

--Analysis Jobs
--*
-- Job1: NumVisits monthly
create table kv_asses41(timeval bigint, invoiceno string); -- creating schema kv pair timestamp to invoice
insert into kv_asses41 select distinct (unix_timestamp(invoicedate,'yyyy-MM-dd HH:mm')),invoiceno from
data_cleaned; -- populating table
create table hourlyhits(dayhr string,invoice string); -- segregation table schema
```

```
insert into hourlyhits select concat(to_date(from_unixtime(timeval)),'->',hour(from_unixtime(timeval))),
invoiceno from kv_asses41; -- filling
create table soln41 as select dayhr,count(invoice) from hourlyhits group by dayhr order by dayhr; -- final
result ;

--Job2: TotalCosts monthly
create table kv_asses42(timeval bigint, totalcost float); -- creating schema kv pair timestamp to cost
insert into kv_asses42 select (unix_timestamp(invoicedate,'yyyy-MM-dd HH:mm')),(quantity*unitprice)
from data_cleaned; -- populating table
create table hourlycosts(dayhr string,costs float); -- segregation table schema
insert into hourlycosts select concat(to_date(from_unixtime(timeval)),'->',hour(from_unixtime(timeval))),
totalcost from kv_asses42; -- filling
create table soln42 as select dayhr,sum(costs) from hourlycosts group by dayhr order by dayhr; -- final
result;

--*;

-- end of code

-- results obtained
--** HIVE shell
--

-- Number of visits per hour
--hive> select * from soln41 limit 10;
--OK
--2010-12-01->10  11
--2010-12-01->11  12
--2010-12-01->12  22
--2010-12-01->13  12
--2010-12-01->14  8
--2010-12-01->15  14
--2010-12-01->16  17
--2010-12-01->17  4
--2010-12-01->8   6
--2010-12-01->9   16
--Time taken: 0.142 seconds, Fetched: 10 row(s)
--hive>


--Total costs per hour
--hive> select * from soln42 limit 10;
--OK
--2010-12-01->10  5235.810002326965
--2010-12-01->11  4234.159994512796
--2010-12-01->12  7447.920057356358
--2010-12-01->13  5063.539980441332
--2010-12-01->14  2831.219994932413
```

```
--2010-12-01->15  3587.3099961280823
--2010-12-01->16  8623.14000633359
--2010-12-01->17  613.190004825592
--2010-12-01->8   1383.8100109100342
--2010-12-01->9   7356.389957070351
--Time taken: 0.174 seconds, Fetched: 10 row(s)
--hive>

--*******
```

Pig Codes

```
/* Analysis 4:
Hourly sales activity like numvisits, totalamount per hour of day. */

-- start of code

--raw data load with defined schema
data_raw= LOAD '/home/kali/Hadoop/Local_Datasets/OnlineRetail.txt' USING PigStorage() as
(InvoiceNo:chararray,StockCode:chararray,Description:chararray,Quantity:int,InvoiceDate:Datetime,UnitPri
ce:float,CustomerID:chararray,Country:chararray);
data_cleaned = FILTER data_raw BY (Quantity>=0 AND UnitPrice>=0.0 AND CustomerID!=''); --cleaning
with condition
ctrRawG= GROUP data_raw ALL;  -- grouping raw to count
ctrClnG= GROUP data_cleaned ALL; -- grouping cleaned to count
ctrRaw= FOREACH ctrRawG GENERATE COUNT(data_raw.Quantity);  --generating count raw
ctrCln= FOREACH ctrClnG GENERATE COUNT(data_cleaned.Quantity); -- generating count cleaned
-- dumping to trigger results' calculation
dump ctrRaw --value :: (541909)
dump ctrCln --value :: (397924)
-- cleaned of debris in data

--analysis job
-- Job1 : NumVisits Monthly
kv_asses41 = FOREACH data_cleaned GENERATE InvoiceDate,InvoiceNo; -- kv pair datetime to invoice
kv_asses41NR = DISTINCT kv_asses41; -- removing duplicates
soln41_prx = FOREACH kv_asses41NR GENERATE CONCAT((chararray)GetDay(InvoiceDate),'-
',(chararray)GetMonth(InvoiceDate),'-',(chararray)GetYear(InvoiceDate),'-
>',(chararray)GetHour(InvoiceDate)) as (dayhr:chararray),InvoiceNo; --presolution
soln41_grp = GROUP soln41_prx BY dayhr;  -- group
soln41 = FOREACH soln41_grp GENERATE group as (dayhr:chararray),COUNT(soln41_prx.InvoiceNo);  --
final solution

-- Job3 : TotalCost Monthly
kv_asses42 = FOREACH data_cleaned GENERATE InvoiceDate,(Quantity*UnitPrice) as (TotalCost:float); --
kv pair datetime to totalcost
```

```
soln42_prx = FOREACH kv_asses42 GENERATE CONCAT((chararray)GetDay(InvoiceDate),'-
',(chararray)GetMonth(InvoiceDate),'-',(chararray)GetYear(InvoiceDate),'-
>',(chararray)GetHour(InvoiceDate)) as (dayhr:chararray),TotalCost; --presolution
soln42_g//Spark code for Analysis4: rp = GROUP soln42_prx BY dayhr;  -- group
soln42 = FOREACH soln42_grp GENERATE group as (dayhr:chararray),SUM(soln42_prx.TotalCost);  -- final
solution

--
--Storage
STORE soln41 INTO '/home/kali/Hadoop/Results/pig_results/analysis4/Job1' USING PigStorage();
STORE soln42 INTO '/home/kali/Hadoop/Results/pig_results/analysis4/Job2' USING PigStorage();

-- end of code

/*
** Results Obtained
#**
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis4/Job1/part* | head -n 10
1-2-2011->8	2
1-2-2011->9	3
1-3-2011->8	4
1-3-2011->9	6
1-4-2011->8	2
1-4-2011->9	6
1-6-2011->7	2
1-6-2011->8	1
1-6-2011->9	3
1-7-2011->8	3
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis4/Job2/part* | head -n 10
1-2-2011->8	1110.6699995994568
1-2-2011->9	1342.5699996948242
1-3-2011->8	990.5500040054321
1-3-2011->9	4896.019991397858
1-4-2011->8	609.5900011062622
1-4-2011->9	2001.5900249481201
1-6-2011->7	326.899995803833
1-6-2011->8	1378.4800004959106
1-6-2011->9	1975.2699918746948
1-7-2011->8	961.8799977302551
kali@kali:~$
#**
*** only 10 results printed to avoid cluttering and scrolling outputs***
*/
```

Spark Codes

```
//Spark code for Analysis4:
```

```
// **Hourly sales activity like numvisits, totalamount per hour of day..
//cleaning of visible debris
var data_raw= sc.textFile("hdfs://localhost:9000/assign1/OnlineRetail.txt")
var data_raw_split= data_raw.map(x=>x.split("\t"))
var data_headless=data_raw_split.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1) else iter
}
data_headless.count
var data_cleaned=data_headless.filter{x=> if((x(3).toInt >=0 ) && (x(5).toFloat >=0.0) && (x(6)!=""))true
else false}
data_cleaned.count
// cleaned data of primary debris

//Analysis Job:
//Job1: Numvisits Hourly
var kvp_dt2inv=data_cleaned.map(x=>{(x(4),x(0))}).distinct
var kv_grp41=kvp_dt2inv.map(x=>{(x._1.substring(0,13),x._2)}).groupByKey()
var soln41=kv_grp41.map(x=>{(x._1,x._2.size)}).sortBy(_._1)

//Job2: TotalCosts Hourly
var kvp_dt2cost=data_cleaned.map(x=>{(x(4),(x(3).toInt*x(5).toFloat))})
var kv_grp42=kvp_dt2cost.map(x=>{(x._1.substring(0,13),x._2)}).groupByKey()
var soln42=kv_grp42.map(x=>{(x._1,x._2.sum)}).sortBy(_._1)
//Save operations
soln41.saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis4/Job1")
soln42.saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis4/Job2")

//end of code



// Solutions Obtained
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis4/Job1/part* | head -n 10
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-18 21:23:48,011 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
//(2010-12-01 10,11)
//(2010-12-01 11,12)
//(2010-12-01 12,22)
//(2010-12-01 13,12)
//(2010-12-01 14,8)
//(2010-12-01 15,14)
//(2010-12-01 16,17)
//(2010-12-01 17,4)
//(2010-12-01 8:,6)
//(2010-12-01 9:,16)
//cat: Unable to write to output stream.
//cat: Unable to write to output stream.
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis4/Job2/part* | head -n 10
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
```

```
//2020-11-18 21:24:04,308 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
//(2010-12-01 10,5235.8096)
//(2010-12-01 11,4234.157)
//(2010-12-01 12,7447.928)
//(2010-12-01 13,5063.5415)
//(2010-12-01 14,2831.2178)
//(2010-12-01 15,3587.3083)
//(2010-12-01 16,8623.145)
//(2010-12-01 17,613.19)
//(2010-12-01 8:,1383.8099)
//(2010-12-01 9:,7356.3896)
//cat: Unable to write to output stream.
//cat: Unable to write to output stream.
//kali@kali:~$
//
// results limited to prevent decluttering of screen as a result the cat error is genererated
```

Question 5.

Python Mapper

```
#!usr/bin/python
#       Basket size distribution (Note: Basket size = number of items in a transaction) **
#       (in this questions, we would like to know that, number of transactions by each basket size) **
#       i.e. number of transactions with 3 size, number of transactions with 4 size etc. **
#  start of code
import sys
for line in sys.stdin:
    row=line.strip()    # removeing white spaces
    components=row.split("\t")       # separating individual components
    try:
        if ( "" not in components and int(components[3])>= 0 and float(components[5])>= 0.0 ):   #
filtration -> quality control
            print('%s\t%d'%(components[0],int(components[3])))   # key: invoice ; values : quantity
        else:
            continue   #ignore condition failures -> quality control
    except Exception:
        continue    # ignore filtration failure -> quality control

# end of code
# results mentioned in the reducer
```

Python Reducer

```
#!usr/bin/python
#
#   Analysis 5:
#       Basket size distribution (Note: Basket size = number of items in a transaction) **
#       (in this questions, we would like to know that, number of transactions by each basket size) **
#       i.e. number of transactions with 3 size, number of transactions with 4 size etc. **
#
# ** start of code

import sys
from operator import itemgetter
data_list=[]        # local list
for line in sys.stdin:
   data_line=line.strip()      #remove whitespaces
   elements=data_line.split("\t")  #segregate components
   data_list.append(elements)  # generate local list
if len(data_list) is 0:
   print(" Input Stream Error ... -> Reducer didn't receive any data ")
   sys.exit(1)         # fatal error due to reducer not receiving any data
data_dict={}  # storing all revenue aggregates
for elem in data_list:
   try:
      # test code for input errors
      if elem[0] not in data_dict:
         # new found unique key :: invoice
         data_dict[elem[0]]=int(elem[1])
      else:
         #  already available country key
         data_dict[elem[0]] = data_dict[elem[0]]+int(elem[1])

   except Exception:
      print (" Input Data Error ... -> Dictionary not generated or In-accessable data list ")

sorted_dc= sorted(data_dict.items(),key = lambda kv:(kv[1],kv[0]),reverse=True)
for unit in sorted_dc:
   print('%s\t%d'%unit)
#

# ** end of code

##
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign5.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign5.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign5.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign5.py" -input /assign1/OnlineRetail.txt -
output /assign1/pythonMR_jobs/analysis5pmr
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis5pmr/part* |head -n 10
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-11-16 10:50:11,714 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
#581483  80995
#541431  74215
#556917  15049
#563076  14730
#572035  13392
#567423  12572
#578841  12540
#552883  12266
#563614  12196
#562439  11848
#cat: Unable to write to output stream.
#kali@kali:~$

#***** note that for job the viewed results are limited to top 10 outputs to avoid printing all 18536
results ****
# ** this restriction imposed to limit output generated a cat error from the resulting stream
```

Hive Codes

```
-- Analysis 5:
-- ** Basket size distribution (Note: Basket size = number of items in a transaction) *
-- *  (in this questions, we would like to know that, number of transactions by each basket size *
-- *   i.e. number of transactions with 3 size, number of transactions with 4 size etc.)  **

-- start of codes
-- one time jobs
-- **## Please avoid lines here on forward if retaildb is available in 'show databases' command and
contains data                  ##**
-- **## if avoiding these lines till line 25 directly jump to codes in analysis section of the code which next
to this section      ##**
create database RetailDB;
use RetailDB;
-- table creation
```

```
create table data_raw_headless (InvoiceNo string,StockCode string,Description string,Quantity
int,InvoiceDate string,UnitPrice float,CustomerID string,Country string) row format delimited fields
terminated by '\t' lines terminated by '\n' tblproperties("skip.header.line.count"="1");
load data inpath 'hdfs://localhost:9000/user/hive/warehouse/OnlineRetail.txt' into table
data_raw_headless;
--check number of datarows
select count (stockcode) from data_raw_headless1 where stockcode!='';
-- result 541909
--cleaning of debris
create table data_cleaned as select * from data_raw_headless1
where (quantity>0 AND unitprice>=0.0 AND customerid != '');
--confirmation and count of cleaned data
show tables;
select count (stockcode) from data_cleaned where stockcode!='';
-- result 397924
-- cleaned of debris

--Analysis Jobs
--*
-- Job: Basket Size
create table kv_asses5 as select (invoiceno),(quantity) from data_cleaned;   -- kv pair invoiceno to quantity
create table analysis5result(invoiceno string,basket_size int); -- creating table for calculating results
insert into analysis5result select invoiceno,sum(quantity) as basket from kv_asses5 group by invoiceno
order by basket desc; -- populating table ;
--*;

-- end of code

-- results obtained
--** HIVE shell
--

--hive> select * from analysis5result limit 10;
--OK
--581483  80995
--541431  74215
--556917  15049
--563076  14730
--572035  13392
--567423  12572
--578841  12540
--552883  12266
--563614  12196
--562439  11848
--Time taken: 0.155 seconds, Fetched: 10 row(s)
--hive>
--
```

```
--***** note that for job the viewed results are limited to top 10 outputs to avoid printing all 18536 results
****
```

Pig Codes

```
/* Analysis 5:
Basket size distribution (Note: Basket size = number of items in a transaction)
(in this questions, we would like to know that, number of transactions by each basket size
i.e. number of transactions with 3 size, number of transactions with 4 size etc.)*/
-- start of code

--raw data load with defined schema
data_raw= LOAD '/home/kali/Hadoop/Local_Datasets/OnlineRetail.txt' USING PigStorage() as
(InvoiceNo:chararray,StockCode:chararray,Description:chararray,Quantity:int,InvoiceDate:Datetime,UnitPric
e:float,CustomerID:chararray,Country:chararray);
data_cleaned = FILTER data_raw BY (Quantity>=0 AND UnitPrice>=0.0 AND CustomerID!=''); --cleaning
with condition
ctrRawG= GROUP data_raw ALL;  -- grouping raw to count
ctrClnG= GROUP data_cleaned ALL; -- grouping cleaned to count
ctrRaw= FOREACH ctrRawG GENERATE COUNT(data_raw.Quantity);  --generating count raw
ctrCln= FOREACH ctrClnG GENERATE COUNT(data_cleaned.Quantity); -- generating count cleaned
-- dumping to trigger results' calculation
dump ctrRaw --value :: (541909)
dump ctrCln --value :: (397924)
-- cleaned of debris in data

--analysis job
-- Job: Basket Size Distribution
kv_asses5 = FOREACH data_cleaned GENERATE InvoiceNo,Quantity; -- kv pair invoiceno to quantity
kv_asses5grp = GROUP kv_asses5 BY InvoiceNo;  -- grouping
analysis5result = FOREACH kv_asses5grp GENERATE group as
(InvoiceNo:chararray),SUM(kv_asses5.Quantity) as (Basket:int);  -- analysis result
orderedRes5 = ORDER analysis5result BY Basket DESC; -- ordering by basket size
STORE orderedRes5 INTO '/home/kali/Hadoop/Results/pig_results/analysis5/' USING PigStorage();   --
storing

-- end of code

/*
** Results Obtained
#**
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis5/part* | head -n 10
581483  80995
541431  74215
556917  15049
563076  14730
572035  13392
```

```
567423  12572
578841  12540
552883  12266
563614  12196
562439  11848
kali@kali:~$
#**
##   ***** note that for job the viewed results are limited to top 10 outputs to avoid printing all 18536
results ****
*/
```

Spark Codes

```
//Spark code for Analysis5:
// ** Basket size distribution (Note: Basket size = number of items in a transaction) *
// *  (in this questions, we would like to know that, number of transactions by each basket size *
// *   i.e. number of transactions with 3 size, number of transactions with 4 size etc.)  **
//cleaning of visible debris
var data_raw= sc.textFile("hdfs://localhost:9000/assign1/OnlineRetail.txt")
var data_raw_split= data_raw.map(x=>x.split("\t"))
var data_headless=data_raw_split.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1) else iter
}
data_headless.count
var data_cleaned=data_headless.filter{x=> if((x(3).toInt >=0 ) && (x(5).toFloat >=0.0) && (x(6)!=""))true
else false}
data_cleaned.count
// cleaned data of primary debris

//analysis jobs
// Job: Basket Size
var kvp_asses5=data_cleaned.map(x=> {(x(0),x(3).toInt)})  // invoice mapped to quantity
var kvp_reduced=kvp_asses5.reduceByKey(_+_) // reduce to sum of quantity
var soln5=kvp_reduced.sortBy(_._2,false) // sorting and final solution
soln5.saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis5") // store

//end of code



// Solutions Obtained
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis5/part* | head -n 10
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-15 20:15:51,720 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
//(581483,80995)
//(541431,74215)
//(556917,15049)
//(563076,14730)
```

```
//(572035,13392)
//(567423,12572)
//(578841,12540)
//(552883,12266)
//(563614,12196)
//(562439,11848)
//cat: Unable to write to output stream.
//cat: Unable to write to output stream.
//kali@kali:~$

//
//***** note that for job the viewed results are limited to top 10 outputs to avoid printing all 18536 results
****
// ** this restriction imposed to limit output generated a cat error from the resulting stream
```

Question 6.

Python Mapper

```
#!usr/bin/python
#
#   Analysis 6:
#       Top 20 Items sold by frequency
#  start of code
import sys
for line in sys.stdin:
    row=line.strip()    # removeing white spaces
    components=row.split("\t")      # separating individual components
    try:
        if ( "" not in components and int(components[3])>= 0 and float(components[5])>= 0.0 ):   #
filtration -> quality control
            print('%s\t%d'%(components[1]+'->'+components[2],int(components[3])))   # key: item ; values :
quantity
        else:
            continue   #ignore condition failures -> quality control
    except Exception:
        continue    # ignore filtration failure -> quality control

# end of code
# results mentioned in the reducer
```

Python Reducer

```
#!usr/bin/python
#   Analysis 6:
#       Top 20 Items sold by frequency
#
# ** start of code

import sys
from operator import itemgetter
data_list=[]        # local list
for line in sys.stdin:
    data_line=line.strip()      #remove whitespaces
    elements=data_line.split("\t")  #segregate components
    data_list.append(elements)  # generate local list
if len(data_list) is 0:
    print(" Input Stream Error ... -> Reducer didn't receive any data ")
    sys.exit(1)         # fatal error due to reducer not receiving any data
data_dict={}  # storing all revenue aggregates
for elem in data_list:
    try:
        # test code for input errors
        if elem[0] not in data_dict:
            # new found unique key :: invoice
            data_dict[elem[0]]=int(elem[1])
        else:
            #  already available country key
            data_dict[elem[0]] = data_dict[elem[0]]+int(elem[1])

    except Exception:
        print (" Input Data Error ... -> Dictionary not generated or In-accessable data list ")

sorted_dc= sorted(data_dict.items(),key = lambda kv:(kv[1],kv[0]),reverse=True)     # sorting  by
descending order
count=0   # counter
for unit in sorted_dc:   # print top 20
    if count<20:
        print('%s\t%d'%unit)
        count=count+1
    else: break
#

# ** end of code

##
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign6.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign6.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign6.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign6.py" -input /assign1/OnlineRetail.txt -output
/assign1/pythonMR_jobs/analysis6pmr
# results obtained
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis6pmr/part*
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-11-16 11:41:10,404 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
#23843->"PAPER CRAFT , LITTLE BIRDIE"    80995
#23166->MEDIUM CERAMIC TOP STORAGE JAR   77916
#84077->WORLD WAR 2 GLIDERS ASSTD DESIGNS        54415
#85099B->JUMBO BAG RED RETROSPOT 46181
#85123A->WHITE HANGING HEART T-LIGHT HOLDER      36725
#84879->ASSORTED COLOUR BIRD ORNAMENT    35362
#21212->PACK OF 72 RETROSPOT CAKE CASES  33693
#22197->POPCORN HOLDER   30931
#23084->RABBIT NIGHT LIGHT       27202
#22492->MINI PAINT SET VINTAGE   26076
#22616->PACK OF 12 LONDON TISSUES        25345
#21977->PACK OF 60 PINK PAISLEY CAKE CASES       24264
#17003->BROCADE RING PURSE       22963
#22178->VICTORIAN GLASS HANGING T-LIGHT  22433
#15036->ASSORTED COLOURS SILK FAN        21876
#21915->RED  HARMONICA IN BOX    20975
#22386->JUMBO BAG PINK POLKADOT  20165
#22197->SMALL POPCORN HOLDER     18252
#20725->LUNCH BAG RED RETROSPOT  17697
#84991->60 TEATIME FAIRY CAKE CASES      17689
#kali@kali:~$
#
#**
```

Hive Codes

```
-- Analysis 6:
-- Top 20 Items sold by frequency

-- start of codes
-- one time jobs
-- **## Please avoid lines here on forward if retaildb is available in 'show databases' command and
contains data                  ##**
```

```
-- **## if avoiding these lines till line 23 directly jump to codes in analysis section of the code which next
to this section      ##**
create database RetailDB;
use RetailDB;
-- table creation
create table data_raw_headless (InvoiceNo string,StockCode string,Description string,Quantity
int,InvoiceDate string,UnitPrice float,CustomerID string,Country string) row format delimited fields
terminated by '\t' lines terminated by '\n' tblproperties("skip.header.line.count"="1");
load data inpath 'hdfs://localhost:9000/user/hive/warehouse/OnlineRetail.txt' into table
data_raw_headless;
--check number of datarows
select count (stockcode) from data_raw_headless1 where stockcode!='';
-- result 541909
--cleaning of debris
create table data_cleaned as select * from data_raw_headless1
where (quantity>0 AND unitprice>=0.0 AND customerid != '');
--confirmation and count of cleaned data
show tables;
select count (stockcode) from data_cleaned where stockcode!='';
-- result 397924
-- cleaned of debris

--Analysis Jobs
--*
-- Job: Top 20 items sold by frequency
create table kv_asses6 (items string,quantity int); -- kv pair items to quantity
insert into kv_asses6 select concat(stockcode,'->',description),(quantity) from data_cleaned;   --
populating data
create table analysis6result(items string,frequency int); -- creating table for calculating results
insert into analysis6result select items,sum(quantity) as freq from kv_asses6 group by items order by freq
desc; -- populating table
create table analysis6T20 row format delimited fields terminated by '\t' stored as textfile
as select * from analysis6result order by frequency desc limit 20; --final result ;
--*;

-- end of code

-- results obtained
--** HIVE shell
--

--hive> select * from analysis6t20;
--OK
--23843->"PAPER CRAFT , LITTLE BIRDIE"    80995
--23166->MEDIUM CERAMIC TOP STORAGE JAR   77916
--84077->WORLD WAR 2 GLIDERS ASSTD DESIGNS        54415
--85099B->JUMBO BAG RED RETROSPOT 46181
--85123A->WHITE HANGING HEART T-LIGHT HOLDER      36725
```

```
--84879->ASSORTED COLOUR BIRD ORNAMENT    35362
--21212->PACK OF 72 RETROSPOT CAKE CASES  33693
--22197->POPCORN HOLDER   30931
--23084->RABBIT NIGHT LIGHT       27202
--22492->MINI PAINT SET VINTAGE   26076
--22616->PACK OF 12 LONDON TISSUES        25345
--21977->PACK OF 60 PINK PAISLEY CAKE CASES       24264
--17003->BROCADE RING PURSE       22963
--22178->VICTORIAN GLASS HANGING T-LIGHT  22433
--15036->ASSORTED COLOURS SILK FAN        21876
--21915->RED  HARMONICA IN BOX    20975
--22386->JUMBO BAG PINK POLKADOT  20165
--22197->SMALL POPCORN HOLDER     18252
--20725->LUNCH BAG RED RETROSPOT  17697
--84991->60 TEATIME FAIRY CAKE CASES      17689
--Time taken: 0.153 seconds, Fetched: 20 row(s)
--hive>
--
-- ****
--
```

Pig Codes

```
/* Analysis 6:
Top 20 Items sold by frequency*/
-- start of code

--raw data load with defined schema
data_raw= LOAD '/home/kali/Hadoop/Local_Datasets/OnlineRetail.txt' USING PigStorage() as
(InvoiceNo:chararray,StockCode:chararray,Description:chararray,Quantity:int,InvoiceDate:Datetime,Unit
Price:float,CustomerID:chararray,Country:chararray);
data_cleaned = FILTER data_raw BY (Quantity>=0 AND UnitPrice>=0.0 AND CustomerID!=''); --
cleaning with condition
ctrRawG= GROUP data_raw ALL;  -- grouping raw to count
ctrClnG= GROUP data_cleaned ALL; -- grouping cleaned to count
ctrRaw= FOREACH ctrRawG GENERATE COUNT(data_raw.Quantity);  --generating count raw
ctrCln= FOREACH ctrClnG GENERATE COUNT(data_cleaned.Quantity); -- generating count cleaned
-- dumping to trigger results' calculation
dump ctrRaw --value :: (541909)
dump ctrCln --value :: (397924)
-- cleaned of debris in data

--analysis job
-- Job: Top 20 items sold by frequency
kv_asses6 = FOREACH data_cleaned GENERATE CONCAT(StockCode,'->',Description) as
(Item:chararray),(Quantity) as (Quantity:int);  --- kv group item to quantity
kv_asses6grp = GROUP kv_asses6 BY Item; -- grouping by item
```

```
analysis6res = FOREACH kv_asses6grp GENERATE group as (Items:chararray),SUM(kv_asses6.Quantity)
as (Frequency:int);
orderedRes = ORDER analysis6res BY Frequency DESC;    -- ordering
result6 = LIMIT orderedRes 20; -- limiting
STORE result6 INTO '/home/kali/Hadoop/Results/pig_results/analysis6/' USING PigStorage();   --
storing

-- end of code

/*
** Results Obtained
#**
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis6/part*
23843->"PAPER CRAFT , LITTLE BIRDIE"    80995
23166->MEDIUM CERAMIC TOP STORAGE JAR   77916
84077->WORLD WAR 2 GLIDERS ASSTD DESIGNS        54415
85099B->JUMBO BAG RED RETROSPOT 46181
85123A->WHITE HANGING HEART T-LIGHT HOLDER      36725
84879->ASSORTED COLOUR BIRD ORNAMENT    35362
21212->PACK OF 72 RETROSPOT CAKE CASES  33693
22197->POPCORN HOLDER   30931
23084->RABBIT NIGHT LIGHT       27202
22492->MINI PAINT SET VINTAGE   26076
22616->PACK OF 12 LONDON TISSUES        25345
21977->PACK OF 60 PINK PAISLEY CAKE CASES       24264
17003->BROCADE RING PURSE       22963
22178->VICTORIAN GLASS HANGING T-LIGHT  22433
15036->ASSORTED COLOURS SILK FAN        21876
21915->RED  HARMONICA IN BOX    20975
22386->JUMBO BAG PINK POLKADOT  20165
22197->SMALL POPCORN HOLDER     18252
20725->LUNCH BAG RED RETROSPOT  17697
84991->60 TEATIME FAIRY CAKE CASES      17689
kali@kali:~$
#**
*/
```

Spark Codes

```
// Analysis 6:
// Top 20 Items sold by frequency

//cleaning of visible debris
var data_raw= sc.textFile("hdfs://localhost:9000/assign1/OnlineRetail.txt")
var data_raw_split= data_raw.map(x=>x.split("\t"))
var data_headless=data_raw_split.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1)
else iter }
```

```
data_headless.count
var data_cleaned=data_headless.filter{x=> if((x(3).toInt >=0 ) && (x(5).toFloat >=0.0) &&
(x(6)!=""))true else false}
data_cleaned.count
// cleaned data of primary debris

//analysis job
// Job: Top 20 items sold by frequency
var kvp_asses6=data_cleaned.map(x=>{(x(1)+"->"+x(2),x(3).toInt)}) // mapping item type to quantity
var kvp_reduced=kvp_asses6.reduceByKey(_+_) //reduce to freq
var sort_res6=kvp_reduced.sortBy(_._2,false) //sort result by frequency
var soln6=sort_res6.take(20) //final result
sc.parallelize(soln6).saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis6")  // save

//end of code



// Solutions Obtained
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis6/part*
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-15 21:31:52,078 WARN util.NativeCodeLoader: Unable to load native-hadoop library for
your platform... using builtin-java classes where applicable
//(23843->"PAPER CRAFT , LITTLE BIRDIE",80995)
//(23166->MEDIUM CERAMIC TOP STORAGE JAR,77916)
//(84077->WORLD WAR 2 GLIDERS ASSTD DESIGNS,54415)
//(85099B->JUMBO BAG RED RETROSPOT,46181)
//(85123A->WHITE HANGING HEART T-LIGHT HOLDER,36725)
//(84879->ASSORTED COLOUR BIRD ORNAMENT,35362)
//(21212->PACK OF 72 RETROSPOT CAKE CASES,33693)
//(22197->POPCORN HOLDER,30931)
//(23084->RABBIT NIGHT LIGHT,27202)
//(22492->MINI PAINT SET VINTAGE ,26076)
//(22616->PACK OF 12 LONDON TISSUES ,25345)
//(21977->PACK OF 60 PINK PAISLEY CAKE CASES,24264)
//(17003->BROCADE RING PURSE ,22963)
//(22178->VICTORIAN GLASS HANGING T-LIGHT,22433)
//(15036->ASSORTED COLOURS SILK FAN,21876)
//(21915->RED  HARMONICA IN BOX ,20975)
//(22386->JUMBO BAG PINK POLKADOT,20165)
//(22197->SMALL POPCORN HOLDER,18252)
//(20725->LUNCH BAG RED RETROSPOT,17697)
//(84991->60 TEATIME FAIRY CAKE CASES,17689)
//kali@kali:~$
//
//*****
//
```

Question 7.

Python Mapper

```
#!usr/bin/python
#
#   Analysis 7:
# ** Customer Lifetime Value distribution by intervals of 1000's (Customer Life time Value = total spend
by customer in his/her tenure with the company) *
# * (In this question, we would like to calculate how many customers with CLV between 1-1000, 1000-
2000 etc.). *
# * Please note that we don't want calculate bins manually and it required to create bins dynamically. **
#
#  start of code
import sys
for line in sys.stdin:
    row=line.strip()    # removeing white spaces
    components=row.split("\t")      # separating individual components
    try:
        if ( "" not in components and int(components[3])>= 0 and float(components[5])>= 0.0 ):   # filtration
-> quality control
            print('%s\t%f'%(components[6],int(components[3])*float(components[5])))   # key: customerid ;
values : totalquantity
        else:
            continue   #ignore condition failures -> quality control
    except Exception:
        continue    # ignore filtration failure -> quality control

# end of code
# results mentioned in the reducer
```

Python Reducer

```
#!usr/bin/python
#
#   Analysis 7:
# ** Customer Lifetime Value distribution by intervals of 1000's (Customer Life time Value = total spend
by customer in his/her tenure with the company) *
# * (In this question, we would like to calculate how many customers with CLV between 1-1000, 1000-
2000 etc.). *
# * Please note that we don't want calculate bins manually and it required to create bins dynamically. **
#
# ** start of code

import sys
```

```
from operator import itemgetter
data_list=[]        # local list
for line in sys.stdin:
  data_line=line.strip()      #remove whitespaces
  elements=data_line.split("\t")  #segregate components
  data_list.append(elements)  # generate local list

if len(data_list) is 0:
  print(" Input Stream Error ... -> Reducer didn't receive any data ")
  sys.exit(1)         # fatal error due to reducer not receiving any data
data_dict={}  # storing all revenue aggregates
for elem in data_list:
  try:
    # test code for input errors
    if elem[0] not in data_dict:
      # new found unique key :: invoice
      data_dict[elem[0]]=float(elem[1])
    else:
      #  already available country key
      data_dict[elem[0]] = data_dict[elem[0]]+float(elem[1])

  except Exception:
    print (" Input Data Error ... -> Dictionary not generated or In-accessable data list ")

collect_dict={}
for an_item in data_dict:
  if (int(data_dict[an_item])/1000)*1000 not in collect_dict:
    collect_dict[(int(data_dict[an_item])/1000)*1000]=[data_dict[an_item]]
  else: collect_dict[(int(data_dict[an_item])/1000)*1000].append(data_dict[an_item])

for unit in sorted(collect_dict):
  print ('%d\t%d\t%d'%(unit,unit+1000,len(collect_dict[unit])))

#

# ** end of code

##
# Operation using Hadoop MapReduce core ->
# command operated on complete file OnlineRetail.txt
# hdfs operative command
#hadoop jar /$HADOOP_HOME/share/hadoop/tools/lib/hadoop-streaming-3.3.0.jar -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign7.py -mapper "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/MapAsign7.py" -file
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign7.py -reducer "python
/home/kali/KomodoIDE/Komodo_jobs/Assign1/RedAsign7.py" -input /assign1/OnlineRetail.txt -output
/assign1/pythonMR_jobs/analysis7pmr
# results obtained
```

```
#**..
#kali@kali:~$ hdfs dfs -cat /assign1/pythonMR_jobs/analysis7pmr/part* | head -n 10
#Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
#2020-11-16 21:02:48,834 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
#0      1000    2671
#1000   2000    765
#2000   3000    347
#3000   4000    182
#4000   5000    99
#5000   6000    66
#6000   7000    46
#7000   8000    26
#8000   9000    19
#9000   10000   14
#kali@kali:~$

#***** note that for job the viewed results are limited to top 10 outputs to avoid printing all results
max(cvl)=280206.03  ****
```

Hive Codes

```
-- Analysis 7:
-- ** Customer Lifetime Value distribution by intervals of 1000’s (Customer Life time Value = total spend
by customer in his/her tenure with the company) *
-- * (In this question, we would like to calculate how many customers with CLV between 1-1000, 1000-
2000 etc.). *
-- * Please note that we don’t want calculate bins manually and it required to create bins dynamically. **

-- start of codes
-- one time jobs
-- **## Please avoid lines here on forward if retaildb is available in 'show databases' command and
contains data                    ##**
-- **## if avoiding these lines till line 23 directly jump to codes in analysis section of the code which next
to this section       ##**
create database RetailDB;
use RetailDB;
-- table creation
create table data_raw_headless (InvoiceNo string,StockCode string,Description string,Quantity
int,InvoiceDate string,UnitPrice float,CustomerID string,Country string) row format delimited fields
terminated by '\t' lines terminated by '\n' tblproperties("skip.header.line.count"="1");
load data inpath 'hdfs://localhost:9000/user/hive/warehouse/OnlineRetail.txt' into table
data_raw_headless;
--check number of datarows
select count (stockcode) from data_raw_headless1 where stockcode!='';
-- result 541909
```

```
--cleaning of debris
create table data_cleaned as select * from data_raw_headless1
where (quantity>0 AND unitprice>=0.0 AND customerid != '');
--confirmation and count of cleaned data
show tables;
select count (stockcode) from data_cleaned where stockcode!='';
-- result 397924
-- cleaned of debris

--Analysis Jobs
--*
-- Job: Customer Lifetime Value distribution by intervals of 1000's
create table kv_asses7(customerid string,totalcost float);  -- table for calculation of customer lifetime value
insert into kv_asses7  select (customerid),sum(quantity*unitprice) as clv from data_cleaned group by
customerid order by clv; -- population of table
create table analysis7collect(limit int,cvl float);   -- table to collect lower limit to cvl
insert into analysis7collect select (int(totalcost/1000))*1000,totalcost from kv_asses7;  -- populating table
create table analysis7result as select limit,(limit+1000),count(cvl) from analysis7collect
where cvl>limit and cvl<limit+1000 group by limit order by limit;  -- final result;
--*;

-- end of code

-- results obtained
--** HIVE shell
--

--hive> select * from analysis7result limit 10;
--OK
--0       1000    2670
--1000    2000    765
--2000    3000    347
--3000    4000    182
--4000    5000    99
--5000    6000    66
--6000    7000    46
--7000    8000    26
--8000    9000    19
--9000    10000   14
--Time taken: 0.172 seconds, Fetched: 10 row(s)
--hive>
--
--***** note that for job the viewed results are limited to top 10 outputs to avoid printing all results
max(cvl)=280206.03  ****
```

Pig Codes

```
/* Analysis 7:
Customer Lifetime Value distribution by intervals of 1000's
(Customer Life time Value = total spend by customer in his/her tenure with the company)
(In this question, we would like to calculate how many customers with CLV between 1-1000, 1000-
2000 etc.).
Please note that we don't want calculate bins manually and it required to create bins dynamically.*/
-- start of code

--raw data load with defined schema
data_raw= LOAD '/home/kali/Hadoop/Local_Datasets/OnlineRetail.txt' USING PigStorage() as
(InvoiceNo:chararray,StockCode:chararray,Description:chararray,Quantity:int,InvoiceDate:Datetime,Unit
Price:float,CustomerID:chararray,Country:chararray);
data_cleaned = FILTER data_raw BY (Quantity>=0 AND UnitPrice>=0.0 AND CustomerID!=''); --
cleaning with condition
ctrRawG= GROUP data_raw ALL;  -- grouping raw to count
ctrClnG= GROUP data_cleaned ALL; -- grouping cleaned to count
ctrRaw= FOREACH ctrRawG GENERATE COUNT(data_raw.Quantity);  --generating count raw
ctrCln= FOREACH ctrClnG GENERATE COUNT(data_cleaned.Quantity); -- generating count cleaned
-- dumping to trigger results' calculation
dump ctrRaw --value :: (541909)
dump ctrCln --value :: (397924)
-- cleaned of debris in data

--analysis job
-- Job: Customer Lifetime Value distribution by intervals of 1000's
kv_asses7 = FOREACH data_cleaned GENERATE CustomerID as (CustomerID:chararray),
UnitPrice*Quantity as (TotalCost:float); -- for calculating costs
kv_asses7grp = GROUP kv_asses7 BY CustomerID;
asses7collect = FOREACH kv_asses7grp GENERATE group as
(CustomerID:chararray),SUM(kv_asses7.TotalCost) as (CLV:float);
asses7res = FOREACH asses7collect GENERATE ROUND(CLV)/1000*1000 as (LoValue:int),CLV;
asses7res_grp = GROUP asses7res BY LoValue;
asses7final = FOREACH asses7res_grp GENERATE group as (LoValue:int),(group + 1000) as
(HiValue:int),COUNT(asses7res.CLV) as (Frequency:int);
order7res = ORDER asses7final BY LoValue;
STORE order7res INTO '/home/kali/Hadoop/Results/pig_results/analysis7/' USING PigStorage();   --
storing

-- end of code

/*
** Results Obtained
#**
kali@kali:~$ cat /home/kali/Hadoop/Results/pig_results/analysis7/part* | head -n 10
0       1000    2671
```

```
1000    2000    765
2000    3000    347
3000    4000    182
4000    5000    99
5000    6000    66
6000    7000    46
7000    8000    26
8000    9000    19
9000    10000   14
kali@kali:~$
#**
## ***** note that for job the viewed results are limited to top 10 outputs to avoid printing all results
max(cvl)=280206.03  ****
*/
```

Spark Codes

```
// Analysis 7
// ** Customer Lifetime Value distribution by intervals of 1000's (Customer Life time Value = total spend
by customer in his/her tenure with the company) *
// * (In this question, we would like to calculate how many customers with CLV between 1-1000, 1000-
2000 etc.). *
// * Please note that we don't want calculate bins manually and it required to create bins dynamically. **

//Start of Code
//cleaning of visible debris
var data_raw= sc.textFile("hdfs://localhost:9000/assign1/OnlineRetail.txt")
var data_raw_split= data_raw.map(x=>x.split("\t"))
var data_headless=data_raw_split.mapPartitionsWithIndex { (idx, iter) => if (idx == 0) iter.drop(1) else iter
}
data_headless.count
var data_cleaned=data_headless.filter{x=> if((x(3).toInt >=0 ) && (x(5).toFloat >=0.0) && (x(6)!=""))true
else false}
data_cleaned.count
// cleaned data of primary debris

// analysis job
var kvp_asses7=data_cleaned.map(x=> {(x(6),x(3).toInt * x(5).toFloat)}) // primary collect
var kvp_clv=kvp_asses7.reduceByKey(_+_) //calculating clv
var lolimit_cvl=kvp_clv.map(x=>{((x._2.toInt/1000)*1000,x._2)}) // limit specified map
var grouped_cvl=lolimit_cvl.sortByKey().groupByKey()    // sort and group
var soln7=grouped_cvl.map(x=>{(x._1,x._1.toInt +1000,x._2.size)}).sortBy(_._1) // final result sorted
soln7.saveAsTextFile("hdfs://localhost:9000/assign1/spark_jobs/analysis7")  //storage

//end of code
```

```
// Solutions Obtained
//kali@kali:~$ hdfs dfs -cat /assign1/spark_jobs/analysis7/part* | head -n 10
//Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
//2020-11-16 02:43:12,748 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
//(0,1000,2671)
//(1000,2000,765)
//(2000,3000,347)
//(3000,4000,182)
//(4000,5000,99)
//(5000,6000,66)
//(6000,7000,46)
//(7000,8000,26)
//(8000,9000,19)
//(9000,10000,14)
//cat: Unable to write to output stream.
//kali@kali:~$

//***** note that for job the viewed results are limited to top 10 outputs to avoid printing all results
max(cvl)=280206.03  ****
// ** this restriction imposed to limit output generated a cat error from the resulting stream
```

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.

All codes pertaining to this project is available on https://github.com/WolfDev8675/RepoSJX7/tree/Assign1

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/