****

**Developed By: Manav Patadia**

**Registration Num.: PGDDSPJULY2020/04**

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

Case Study 1

## Retail Sales

## Acknowledgement

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

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

## Table of Contents

[Retail Sales 1](file:///G:\PGD_class\Retail-Sales-Big-Data\Retail%20Sales%20Big%20Data%20Project%20-%20Copy.docx#_Toc56559452)

[Acknowledgement 2](#_Toc56559453)

[Table of Contents 3](#_Toc56559454)

[Objective & Purpose 4](#_Toc56559455)

[Functionality 5](#_Toc56559456)

[Hardware & Software Requirements 6](#_Toc56559457)

[Why Hadoop 7](#_Toc56559458)

[Big Data Tools 8](#_Toc56559459)

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

[Data Preparation 10](#_Toc56559461)

[Question 1 11](#_Toc56559462)

[Solution 1: 11](#_Toc56559463)

[Question 2 15](#_Toc56559464)

[Solution 2: 15](#_Toc56559465)

[Question 3 39](#_Toc56559466)

[Solution 3: 39](#_Toc56559467)

[Question 4 46](#_Toc56559468)

[Solution 4: 46](#_Toc56559469)

[Question 5 50](#_Toc56559470)

[Solution 5: 50](#_Toc56559471)

[Question 6 54](#_Toc56559472)

[Solution 6: 54](#_Toc56559473)

[Question 7 58](#_Toc56559474)

[Solution 7: 58](#_Toc56559475)

## Objective & Purpose

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

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

## Functionality

1. **Revenue Aggregate By Country for top 5 countries:** Calculate sum of total amount ( unit price \* quantity ) of every transaction in each countries and display top 5 countries having highest revenue.
2. **Sales Metrics like NumCustomers, NumTransactions, AvgNumItems, MinAmtperCustomer, MaxAmtperCustomer and AvgAmtperCustomer by country for top 5 countries: NumCustomers** is count of all unique customers per country and display top 5 countries having highest number of customers. **NumTransaction** is count of all transactions per country and display top 5 countries having highest number of transactions. **AvgNumItems** is average number of items purchased by each customer. **MinAmtperCustomer** is minimum amount spend by each customer per transaction. **MaxAmtperCustomer** is maximum amount spend by each customer per transaction. **AvgAmtperCustomer** is average amount spend by each customer per transaction.
3. **Daily Sales Activity like NumVisits and TotalAmt monthly and quarterly for each year:** Calculate total number of customers visted and total amount of money spent by customers per month and per quarter.
4. **Hourly sales Activity like NumVisits and TotalAmt per hour of day:** Calculate total number of customers visted and total amount of money spent by customers per hour.
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.):** Calculate number of items purchased per each transaction and group transaction with same number of items and count its frequency.
6. **Top 20 Items sold by frequency:** Count number of times each item is sold and display top 20 items that are sold most.
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:** Calculate CLV by adding all total amount spend by each customer and group each CLV in the interval’s of 1000’s.

## Hardware & Software Requirements

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

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

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

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

## Why Hadoop

**1. Scalable**

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

**2. Cost effective**

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

**3. Flexible**

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

**4. Fast**

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

**5. Resilient to failure**

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

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

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

## Big Data Tools

**Pig vs Hive**

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

**Map-Reduce vs Spark**

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

## Future Scope of Big Data

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

## Data Preparation

### Sample Data

HDFS Commands for Input and Output folders:

hdfs dfs -mkdir /RETAIL\_SALES

hdfs dfs -put OnlineRetail.txt /RETAIL\_SALES

hdfs dfs -mkdir /RETAIL\_SALES\_OUTPUT/

hdfs dfs -mkdir /RETAIL\_SALES\_OUTPUT/HIVE/

hdfs dfs -mkdir /RETAIL\_SALES\_OUTPUT/PIG/

hdfs dfs -mkdir /RETAIL\_SALES\_OUTPUT/MAPREDUCE/

hdfs dfs -mkdir /RETAIL\_SALES\_OUTPUT/SPARK/

HIVE Commands for Table Creation:

CREATE DATABASE UC1\_RETAIL;

USE UC1\_RETAIL;

CREATE EXTERNAL TABLE ONLINERETAIL (

INVOICENO INT, STOCKCODE STRING, DESCRIPTION STRING, QUANTITY INT, INVOICEDATE STRING, UNITPRICE DOUBLE, CUSTOMERID STRING, COUNTRY STRING )

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

LOCATION '/RETAIL\_SALES';

## Question 1

### 1. Revenue Aggregate By Country for top 5 countries:

### Calculate sum of total amount ( unit price \* quantity ) of every transaction in each countries and display top 5 countries having highest revenue.

## Solution 1:

### Hive

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/1'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT COUNTRY, ROUND(SUM(UNITPRICE \* QUANTITY),2) AS SUM\_UQ

FROM UC1\_RETAIL.ONLINERETAIL

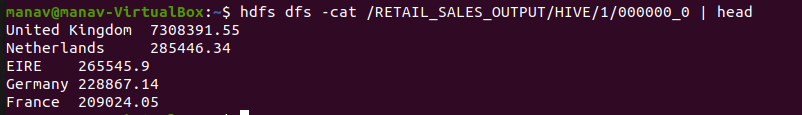
WHERE UNITPRICE >=0 AND QUANTITY >0 AND TRIM(CUSTOMERID) <> ''

GROUP BY COUNTRY

ORDER BY SUM\_UQ DESC

LIMIT 5;

##### Output:



### Pig

##### Pig Commands:

RETAIL = LOAD 'hdfs://localhost:9000/RETAIL\_SALES/' USING PigStorage('\t') AS ( INVOICENO:INT, STOCKCODE:CHARARRAY, DESCRIPTION:CHARARRAY, QUANTITY:INT, INVOICEDATE:CHARARRAY, UNITPRICE:DOUBLE, CUSTOMERID:CHARARRAY, COUNTRY:CHARARRAY );

FILTER\_DATA = FILTER RETAIL BY UNITPRICE >=0 AND QUANTITY >0 AND CUSTOMERID != '';

AMOUNT = FOREACH FILTER\_DATA GENERATE UNITPRICE \* QUANTITY AS (PQ:FLOAT), COUNTRY;

COUNTRY\_GROUP = GROUP AMOUNT BY COUNTRY;

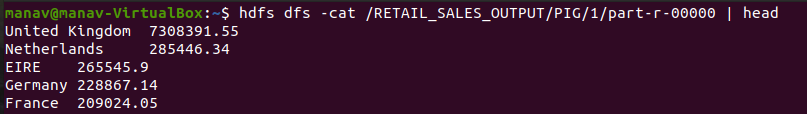
SUM\_AMOUNT = FOREACH COUNTRY\_GROUP GENERATE group, ROUND\_TO(SUM(AMOUNT.PQ),2) AS REVENUE\_AGGREGATE;

SORT\_REVENUE = ORDER SUM\_AMOUNT BY REVENUE\_AGGREGATE DESC;

REVENUE\_LIMIT\_5 = LIMIT SORT\_REVENUE 5;

STORE REVENUE\_LIMIT\_5 INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/1/' USING PigStorage ('\t');

##### Output:



### MapReduce

##### RevenueAgreegate\_mapper1.py

#!usr/bin/python3

import sys

for row in sys.stdin:

col = row.strip().split('\t')

try:

country, unitprice, quantity, custID = col[7], float(col[5]), int(col[3]), col[6].strip()

if( unitprice >=0 and quantity >0 and str(custID) != ''):

print('%s\t%s' % (country, unitprice \* quantity))

except ValueError: pass

##### RevenueAgreegate\_reducer1.py

#!usr/bin/python3

import sys

sumUQ = {}

for row in sys.stdin:

country, UQ = row.strip().split('\t', 1)

UQ = float(UQ)

try:

sumUQ[country] = sumUQ[country] + UQ

except:

sumUQ[country] = UQ

revenueAggregate = sorted(sumUQ.items(), key=lambda x: x[1], reverse=True)

x=0

print("Country\t\t\tRevenue Aggregate")

for i in revenueAggregate:

x = x + 1

if(x<=5):

print(i[0], "\t\t\t",str(round(i[1],2)))

else:

break

|  |
| --- |
|  |

##### Execution Command:

hadoop jar hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/RevenueAgreegate\_mapper1.py -mapper "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/RevenueAgreegate\_mapper1.py" -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/RevenueAgreegate\_reducer1.py -reducer "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/RevenueAgreegate\_reducer1.py" -input /RETAIL\_SALES/OnlineRetail.txt -output /RETAIL\_SALES\_OUTPUT/MAPREDUCE/OUTPUT/1

##### G:\PGD_class\Retail-Sales-Big-Data\OUTPUT\mapreduce_1.PNGOutput:

### Spark

val InputFile = "hdfs://localhost:9000/RETAIL\_SALES/OnlineRetail.txt"

val OutputFile = "hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/SPARK/"

val INVOICENO = 0

val STOCKCODE = 1

val DESCRIPTION = 2

val QUANTITY = 3

val INVOICEDATE = 4

val UNITPRICE = 5

val CUSTOMERID = 6

val COUNTRY = 7

##### Spark Commands:

val OnlineRetail = sc.textFile(InputFile).map(row => row.split("\t"))

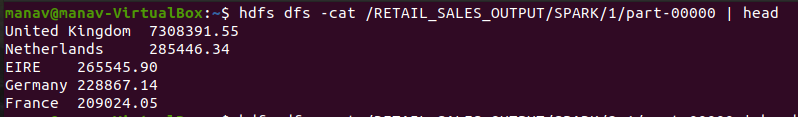
val FilterMap = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(COUNTRY).toString, col(QUANTITY).toInt \* BigDecimal(col(UNITPRICE)).setScale(2, BigDecimal.RoundingMode.HALF\_UP))})

val TotalRevenue = FilterMap.groupByKey().mapValues(sq => (sq.sum)).sortBy(\_.\_2, false)

val RevenueAggregate = sc.parallelize(TotalRevenue.take(5))

RevenueAggregate.map(r => r.productIterator.mkString("\t")).saveAsTextFile(OutputFile+"1")

##### Output:



## Question 2

### 2. Sales Metrics like NumCustomers, NumTransactions, AvgNumItems, MinAmtperCustomer, MaxAmtperCustomer and AvgAmtperCustomer by country for top 5 countries.

## Solution 2:

### Hive

#### NumCustomers: count of all unique customers per country and display top 5 countries having highest number of customers

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/2\_1'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT COUNTRY, COUNT(DISTINCT CUSTOMERID) AS NumCustomers

FROM UC1\_RETAIL.ONLINERETAIL

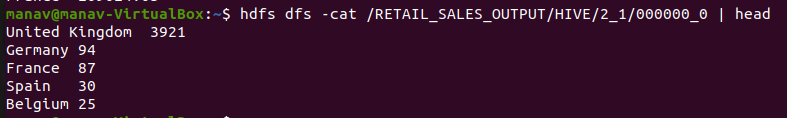
WHERE UNITPRICE >=0 AND QUANTITY >0 AND TRIM(CUSTOMERID) <> ''

GROUP BY COUNTRY

ORDER BY NumCustomers DESC

LIMIT 5;

##### Output:



#### NumTransactions: count of all transactions per country and display top 5 countries having highest number of transactions

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/2\_2'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT COUNTRY, COUNT(DISTINCT INVOICENO) AS NumTransactions

FROM UC1\_RETAIL.ONLINERETAIL

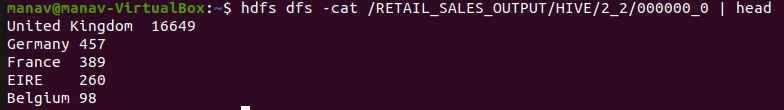
WHERE UNITPRICE >=0 AND QUANTITY >0 AND TRIM(CUSTOMERID) <> ''

GROUP BY COUNTRY

ORDER BY NumTransactions DESC

LIMIT 5;

##### Output:



#### AvgNumItems: is average number of items purchased by each customer

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/2\_3'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT CUSTOMERID, ROUND((SUM(SUM\_Q)/COUNT(INVOICENO)),2) AS AvgNumItems FROM (

SELECT INVOICENO, SUM(QUANTITY) AS SUM\_Q, CUSTOMERID

FROM UC1\_RETAIL.ONLINERETAIL

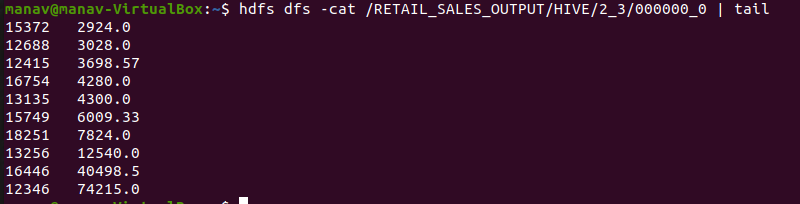
WHERE UNITPRICE >=0 and QUANTITY >0 and TRIM(CUSTOMERID) <> ''

GROUP BY INVOICENO, CUSTOMERID ) RETAIL\_TEMP

GROUP BY CUSTOMERID

ORDER BY AvgNumItems;

##### Output:



#### MinAmtperCustomer: minimum amount spend by each customer per transaction

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/2\_4'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT CUSTOMERID, MIN(SUM\_UQ) AS MinAmtperCustomer FROM (

SELECT INVOICENO, CAST(ROUND(SUM(UNITPRICE \* QUANTITY),2) AS DOUBLE) AS SUM\_UQ, CUSTOMERID

FROM UC1\_RETAIL.ONLINERETAIL

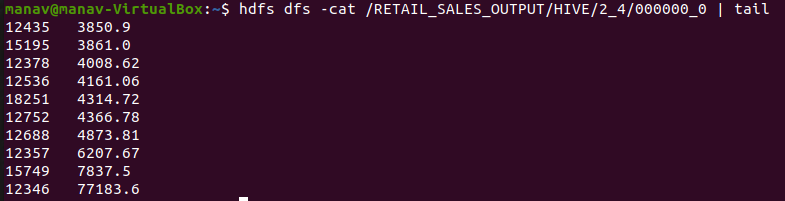
WHERE UNITPRICE >=0 and QUANTITY >0 and TRIM(CUSTOMERID) <> ''

GROUP BY INVOICENO, CUSTOMERID) RETAIL\_TEMP

GROUP BY CUSTOMERID

ORDER BY MinAmtperCustomer;

##### Output:



#### MaxAmtperCustomer: maximum amount spend by each customer per transaction

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/2\_5'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT CUSTOMERID, MAX(SUM\_UQ) AS MaxAmtperCustomer FROM (

SELECT INVOICENO, CAST(ROUND(SUM(UNITPRICE \* QUANTITY),2) AS DOUBLE) AS SUM\_UQ, CUSTOMERID

FROM UC1\_RETAIL.ONLINERETAIL

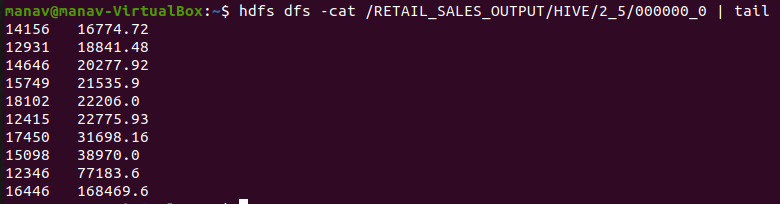
WHERE UNITPRICE >=0 and QUANTITY >0 and TRIM(CUSTOMERID) <> ''

GROUP BY INVOICENO, CUSTOMERID) RETAIL\_TEMP

GROUP BY CUSTOMERID

ORDER BY MaxAmtperCustomer;

##### Output:



#### AvgAmtperCustomer: average amount spend by each customer per transaction

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/2\_6'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT CUSTOMERID, ROUND((SUM(SUM\_UQ)/COUNT(CUSTOMERID)),2) AS AvgAmtperCustomer FROM (

SELECT INVOICENO, CAST(ROUND(SUM(UNITPRICE \* QUANTITY),2) AS DOUBLE) AS SUM\_UQ, CUSTOMERID

FROM UC1\_RETAIL.ONLINERETAIL

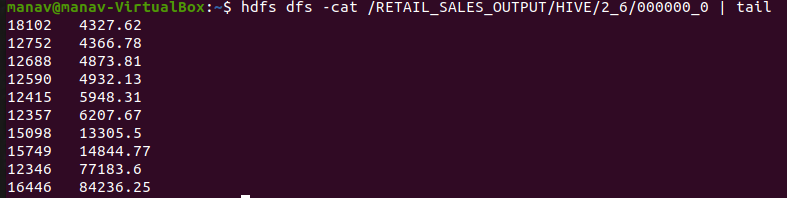
WHERE UNITPRICE >=0 and QUANTITY >0 and TRIM(CUSTOMERID) <> ''

GROUP BY INVOICENO, CUSTOMERID ) RETAIL\_TEMP

GROUP BY CUSTOMERID

ORDER BY AvgAmtperCustomer;

##### Output:



### Pig

RETAIL = LOAD 'hdfs://localhost:9000/RETAIL\_SALES/' USING PigStorage('\t') AS ( INVOICENO:INT, STOCKCODE:CHARARRAY, DESCRIPTION:CHARARRAY, QUANTITY:INT, INVOICEDATE:CHARARRAY, UNITPRICE:DOUBLE, CUSTOMERID:CHARARRAY, COUNTRY:CHARARRAY );

FILTER\_DATA = FILTER RETAIL BY UNITPRICE >=0 AND QUANTITY >0 AND CUSTOMERID != '';

#### NumCustomers

##### Pig Commands:

GROUP\_COUNTRY = GROUP FILTER\_DATA BY COUNTRY;

COUNT\_CUSTOMER\_ID = FOREACH GROUP\_COUNTRY {

DISTINCT\_CUSTID = DISTINCT FILTER\_DATA.CUSTOMERID;

GENERATE group, COUNT(DISTINCT\_CUSTID) AS (CUSTID\_COUNT:LONG);

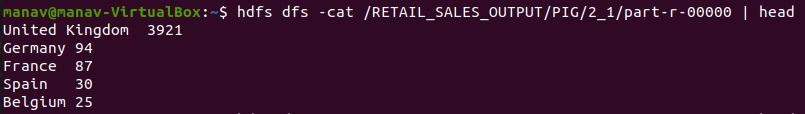
};

SORT\_NUM\_CUSTID = ORDER COUNT\_CUSTOMER\_ID BY CUSTID\_COUNT DESC;

NUM\_CUSTOMERS = LIMIT SORT\_NUM\_CUSTID 5;

STORE NUM\_CUSTOMERS INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/2\_1/' USING PigStorage ('\t');

##### Output:



#### NumTransactions

##### Pig Commands:

GROUP\_COUNTRY = GROUP FILTER\_DATA BY COUNTRY;

COUNT\_INVOICE = FOREACH GROUP\_COUNTRY {

DISTINCT\_TRANSACTIONS = DISTINCT FILTER\_DATA.INVOICENO;

GENERATE group, COUNT(DISTINCT\_TRANSACTIONS) AS (TRANSACTION\_COUNT:LONG);

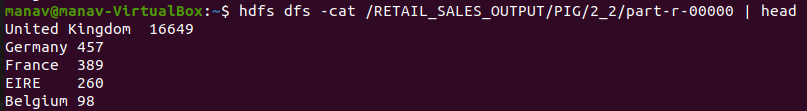
};

ORDER\_NUMTRANSACTIONS = ORDER COUNT\_INVOICE BY TRANSACTION\_COUNT DESC;

NUM\_TRANSACTIONS = LIMIT ORDER\_NUMTRANSACTIONS 5;

STORE NUM\_TRANSACTIONS INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/2\_2/' USING PigStorage ('\t');

##### Output:



#### AvgNumItems

##### Pig Commands:

IC\_GROUP = GROUP FILTER\_DATA BY (INVOICENO, CUSTOMERID);

SUM\_QUANTITY = FOREACH IC\_GROUP {

GENERATE group, group.CUSTOMERID AS (CUSTID:CHARARRAY), SUM(FILTER\_DATA.QUANTITY) AS (SUM\_Q:LONG);

};

CUSTID\_GROUP = GROUP SUM\_QUANTITY BY CUSTID;

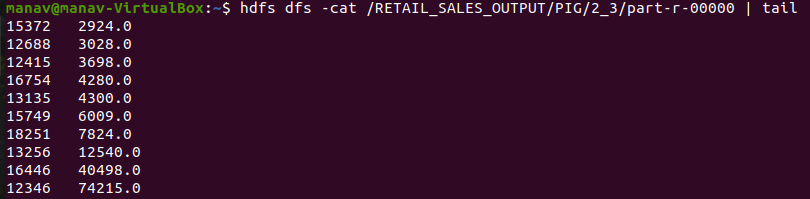
QUANTITY\_AGGREGATE = FOREACH CUSTID\_GROUP GENERATE group, SUM(SUM\_QUANTITY.SUM\_Q) AS (TOTAL\_QUANTITY:LONG), COUNT(SUM\_QUANTITY.CUSTID) AS (COUNT\_QUANTITY:LONG);

AVG = FOREACH QUANTITY\_AGGREGATE GENERATE group, ROUND\_TO(TOTAL\_QUANTITY/COUNT\_QUANTITY,2) AS (AVGNUMITEMS:FLOAT);

SORT\_AVGNUMITEMS = ORDER AVG BY AVGNUMITEMS;

STORE SORT\_AVGNUMITEMS INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/2\_3/' USING PigStorage ('\t');

##### Output:



#### MinAmtperCustomer

##### Pig Commands:

IC\_GROUP = GROUP FILTER\_DATA BY (INVOICENO, CUSTOMERID);

SUM\_PQ = FOREACH IC\_GROUP {

TOTAL\_PQ = FOREACH FILTER\_DATA GENERATE UNITPRICE \* QUANTITY AS (PQ:FLOAT);

GENERATE group, group.CUSTOMERID AS (CUSTID:CHARARRAY), SUM(TOTAL\_PQ.PQ) AS (REVENUE\_AGGREGATE:DOUBLE);

};

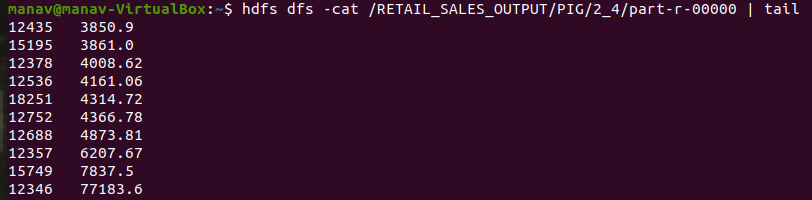
CUSTID\_GROUP = GROUP SUM\_PQ BY CUSTID;

REVENUE = FOREACH CUSTID\_GROUP GENERATE group, ROUND\_TO(MIN(SUM\_PQ.REVENUE\_AGGREGATE),2) AS (MINAMTPERCUSTOMER:DOUBLE);

SORT\_MINAMTPERCUSTOMER = ORDER REVENUE BY MINAMTPERCUSTOMER;

STORE SORT\_MINAMTPERCUSTOMER INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/2\_4/' USING PigStorage ('\t');

##### Output:



#### MaxAmtperCustomer

##### Pig Commands:

IC\_GROUP = GROUP FILTER\_DATA BY (INVOICENO, CUSTOMERID);

SUM\_PQ = FOREACH IC\_GROUP {

TOTAL\_PQ = FOREACH FILTER\_DATA GENERATE UNITPRICE \* QUANTITY AS (PQ:FLOAT);

GENERATE group, group.CUSTOMERID AS (CUSTID:CHARARRAY), SUM(TOTAL\_PQ.PQ) AS (REVENUE\_AGGREGATE:DOUBLE);

};

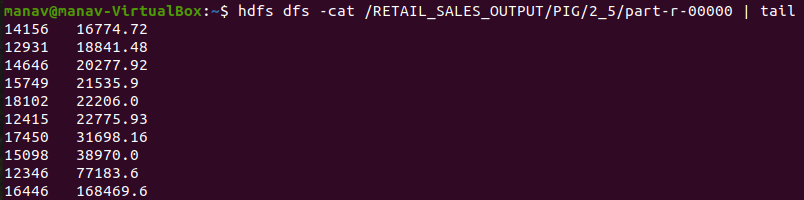
CUSTID\_GROUP = GROUP SUM\_PQ BY CUSTID;

REVENUE = FOREACH CUSTID\_GROUP GENERATE group, ROUND\_TO(MAX(SUM\_PQ.REVENUE\_AGGREGATE),2) AS (MAXAMTPERCUSTOMER:DOUBLE);

SORT\_MAXAMTPERCUSTOMER = ORDER REVENUE BY MAXAMTPERCUSTOMER;

STORE SORT\_MAXAMTPERCUSTOMER INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/2\_5/' USING PigStorage ('\t');

##### Output:



#### AvgAmtperCustomer

##### Pig Commands:

IC\_GROUP = GROUP FILTER\_DATA BY (INVOICENO, CUSTOMERID);

SUM\_PQ = FOREACH IC\_GROUP {

TOTAL\_PQ = FOREACH FILTER\_DATA GENERATE UNITPRICE \* QUANTITY AS (PQ:FLOAT);

GENERATE group, group.CUSTOMERID AS (CUSTID:CHARARRAY), SUM(TOTAL\_PQ.PQ) AS (REVENUE\_AGGREGATE:DOUBLE);

};

CUSTID\_GROUP = GROUP SUM\_PQ BY CUSTID;

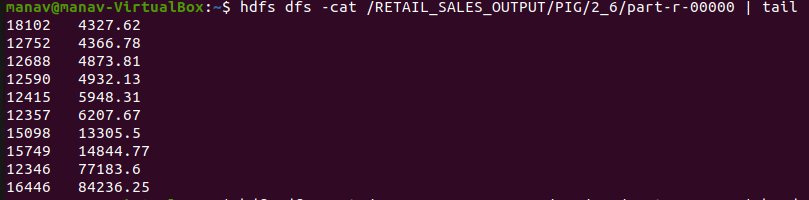
REVENUE = FOREACH CUSTID\_GROUP GENERATE group, SUM(SUM\_PQ.REVENUE\_AGGREGATE) AS (TOTAL\_QUANTITY:DOUBLE), COUNT(SUM\_PQ.CUSTID) AS (COUNT\_QUANTITY:LONG);

AVG = FOREACH REVENUE GENERATE group, ROUND\_TO(TOTAL\_QUANTITY/COUNT\_QUANTITY,2) AS (AVGAMTPERCUSTOMER:DOUBLE);

SORT\_AVGAMTPERCUSTOMER = ORDER AVG BY AVGAMTPERCUSTOMER;

STORE SORT\_AVGAMTPERCUSTOMER INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/2\_6/' USING PigStorage ('\t');

##### Output:



### MapReduce

#### NumCustomers:

##### NumCustomers\_mapper2\_1.py

#!usr/bin/python3

import sys, math

for row in sys.stdin:

col = row.strip().split('\t')

try:

country, unitprice, quantity, custID = col[7], float(col[5]), int(col[3]), col[6].strip()

if( unitprice >=0 and quantity >0 and str(custID) != ''):

print('%s\t%s' % (country, custID))

except ValueError: pass

##### NumCustomers\_reducer2\_1.py

#!usr/bin/python3

import sys, math

from collections import OrderedDict

custCount = {}

input=[]

for row in sys.stdin:

input.append(row)

for row in list(set(input)):

country, custId = row.strip().split('\t',1)

try:

custCount[country] = custCount[country] + 1

except:

custCount[country] = 1

numCustomers = OrderedDict(sorted(custCount.items(), key=lambda x: x[1], reverse=True))

x=0

print("Country\t\t\tNumCustomers")

for (key,value) in numCustomers.items():

x = x + 1

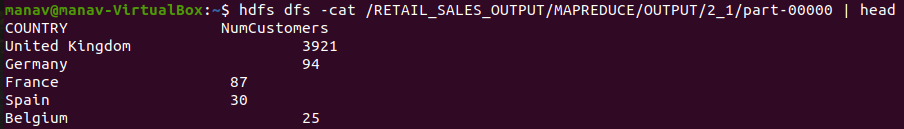
if(x <= 5):

print(key, "\t\t\t", value)

##### Execution Command:

hadoop jar hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumCustomers\_mapper2\_1.py -mapper "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumCustomers\_mapper2\_1.py" -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumCustomers\_reducer2\_1.py -reducer "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumCustomers\_reducer2\_1.py" -input /RETAIL\_SALES/OnlineRetail.txt -output /RETAIL\_SALES\_OUTPUT/MAPREDUCE/OUTPUT/2\_1

##### Output:



#### NumTransactions:

##### NumTransactions\_mapper2\_2.py

#!usr/bin/python3

import sys, math

for row in sys.stdin:

col = row.strip().split('\t')

try:

country, invoiceNum, unitprice, quantity, custID = col[7], col[0].strip(), float(col[5]), int(col[3]), col[6].strip()

if( unitprice >=0 and quantity >0 and str(custID) != ''):

print('%s\t%s' % (country, invoiceNum))

except ValueError: pass

##### NumTransactions\_reducer2\_2.py

#!usr/bin/python3

import sys, math

from collections import OrderedDict

countInvoice = {}

invoice=[]

invoice.append(" ")

input=[]

for row in sys.stdin:

input.append(row)

for row in list(set(input)):

country, invoiceNum = row.strip().split('\t',1)

if(invoiceNum not in invoice):

invoice.append(invoiceNum)

try:

countInvoice[country] = countInvoice[country] + 1

except:

countInvoice[country] = 1

invoice\_count = OrderedDict(sorted(countInvoice.items(), key=lambda x: x[1], reverse=True))

x=0

print("Country\t\t\tNumTransactions")

for (key,value) in invoice\_count.items():

x = x + 1

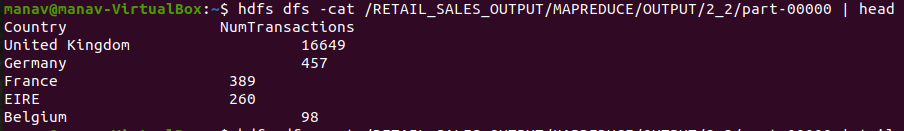
if(x <= 5):

print(key, "\t\t\t", value)

##### Execution Command:

hadoop jar hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumTransactions\_mapper2\_2.py -mapper "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumTransactions\_mapper2\_2.py" -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumTransactions\_reducer2\_2.py -reducer "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumTransactions\_reducer2\_2.py" -input /RETAIL\_SALES/OnlineRetail.txt -output /RETAIL\_SALES\_OUTPUT/MAPREDUCE/OUTPUT/2\_2

##### Output:



#### AvgNumItems:

##### AvgNumItems\_mapper2\_3.py

#!usr/bin/python3

import sys, math

for row in sys.stdin:

col = row.strip().split('\t')

try:

invoiceNum, unitprice, quantity, custID = col[0].strip(), float(col[5]), int(col[3]), col[6].strip()

if( unitprice >=0 and quantity >0 and str(custID) != ''):

print('%s\t%s\t%s' % (invoiceNum, custID, quantity))

except ValueError: pass

##### AvgNumItems\_reducer2\_3.py

#!usr/bin/python3

import sys, math

from collections import OrderedDict

qtyCount = {}

sumQtyCount = {}

qtyAvg = {}

custIdCount = {}

input=[]

for row in sys.stdin:

invoiceNum, custId, qty = row.strip().split('\t',2)

try:

qtyCount[invoiceNum+'\_'+custId] = qtyCount[invoiceNum+'\_'+custId] + int(qty)

except:

qtyCount[invoiceNum+'\_'+custId] = int(qty)

for (key,value) in qtyCount.items():

invoiceNum, custId = key.strip().split('\_',1)

try:

sumQtyCount[custId] = sumQtyCount[custId] + int(value)

except:

sumQtyCount[custId] = int(value)

for (key,value) in qtyCount.items():

invoiceNum, custId = key.strip().split('\_',1)

try:

custIdCount[custId] = custIdCount[custId] + 1

except:

custIdCount[custId] = 1

SortedQtyCount = OrderedDict(sorted(sumQtyCount.items(), key=lambda x: x[0], reverse=True))

SortedCustIdCount = OrderedDict(sorted(custIdCount.items(), key=lambda x: x[0], reverse=True))

for (key,value), (key2,value2) in zip(SortedQtyCount.items(), SortedCustIdCount.items()):

if(key==key2):

value = int(float(value))

value2 = int(float(value2))

qtyAvg[key]=str(round(value/value2,2))

qtyAvg\_sorted = OrderedDict(sorted(qtyAvg.items(), key=lambda x: float(x[1])))

print("Customer ID\t\t\tAvgNumItems")

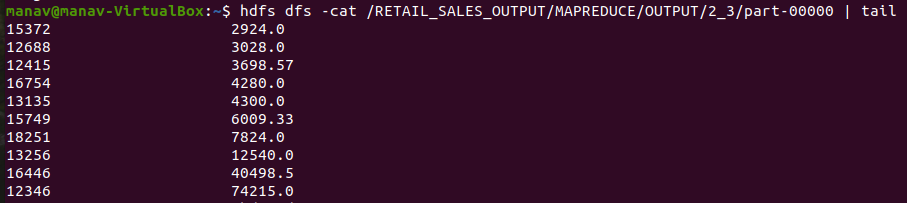
for (key,value) in qtyAvg\_sorted.items():

print(key,'\t\t\t',value)

##### Execution Command:

hadoop jar hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/AvgNumItems\_mapper2\_3.py -mapper "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/AvgNumItems\_mapper2\_3.py" -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/AvgNumItems\_reducer2\_3.py -reducer "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/AvgNumItems\_reducer2\_3.py" -input /RETAIL\_SALES/OnlineRetail.txt -output /RETAIL\_SALES\_OUTPUT/MAPREDUCE/OUTPUT/2\_3

##### Output:



#### MinAmtperCustomer:

##### MinAmtperCustomer\_mapper2\_4.py

#!usr/bin/python3

import sys

for row in sys.stdin:

col = row.strip().split('\t')

try:

invoiceNum, unitprice, quantity, custID = col[0], float(col[5]), int(col[3]), col[6]

if( unitprice >=0 and quantity >0 and str(custID) != ''):

print('%s\t%s\t%s' % (invoiceNum, custID, unitprice\*quantity))

except ValueError: pass

##### MinAmtperCustomer\_reducer2\_4.py

#!usr/bin/python3

import sys, math

from collections import OrderedDict

from itertools import groupby

revenueAggregate = {}

revenueGroup = {}

minAmtPerCustomer = {}

for row in sys.stdin:

invoiceNum, custId, UQ = row.strip().split('\t',2)

try:

revenueAggregate[custId+'\_'+invoiceNum] = revenueAggregate[custId+'\_'+invoiceNum] + float(UQ)

except:

revenueAggregate[custId+'\_'+invoiceNum] = float(UQ)

revenueAggregate\_sorted = OrderedDict(sorted(revenueAggregate.items(), key=lambda x: float(x[0])))

for (key,value) in revenueAggregate\_sorted.items():

custId, invoiceNum = key.split('\_',1)

try:

revenueGroup[custId] = revenueGroup[custId] + ',' + str(value)

except:

revenueGroup[custId] = str(value)

for (key,value) in revenueGroup.items():

temp = value.split(',')

temp = [round(float(i),2) for i in temp]

minAmtPerCustomer[key]=min(temp)

print("CustomerId\t\t\tMinAmtperCustomer")

minAmtPerCustomer\_sorted = OrderedDict(sorted(minAmtPerCustomer.items(), key=lambda x: float(x[1])))

for (key,value) in minAmtPerCustomer\_sorted.items():

print(key,'\t\t\t',str(value))

##### Execution Command:

hadoop jar hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/MinAmtperCustomer\_mapper2\_4.py -mapper "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/MinAmtperCustomer\_mapper2\_4.py" -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/MinAmtperCustomer\_reducer2\_4.py -reducer "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/MinAmtperCustomer\_reducer2\_4.py" -input /RETAIL\_SALES/OnlineRetail.txt -output /RETAIL\_SALES\_OUTPUT/MAPREDUCE/OUTPUT/2\_4

##### G:\PGD_class\Retail-Sales-Big-Data\OUTPUT\mapreduce_2_4.PNGOutput:

#### MaxAmtperCustomer:

##### MaxAmtperCustomer\_mapper2\_5.py

#!usr/bin/python3

import sys

for row in sys.stdin:

col = row.strip().split('\t')

try:

invoiceNum, unitprice, quantity, custID = col[0], float(col[5]), int(col[3]), col[6]

if( unitprice >=0 and quantity >0 and str(custID) != ''):

print('%s\t%s\t%s' % (invoiceNum, custID, unitprice\*quantity))

except ValueError: pass

##### MaxAmtperCustomer\_reducer2\_5.py

#!usr/bin/python3

import sys, math

from collections import OrderedDict

from itertools import groupby

revenueAggregate = {}

revenueGroup = {}

maxAmtPerCustomer = {}

for row in sys.stdin:

invoiceNum, custId, UQ = row.strip().split('\t',2)

try:

revenueAggregate[custId+'\_'+invoiceNum] = revenueAggregate[custId+'\_'+invoiceNum] + float(UQ)

except:

revenueAggregate[custId+'\_'+invoiceNum] = float(UQ)

revenueAggregate\_sorted = OrderedDict(sorted(revenueAggregate.items(), key=lambda x: float(x[0])))

for (key,value) in revenueAggregate\_sorted.items():

custId, invoiceNum = key.split('\_',1)

try:

revenueGroup[custId] = revenueGroup[custId] + ',' + str(value)

except:

revenueGroup[custId] = str(value)

for (key,value) in revenueGroup.items():

temp = value.split(',')

temp = [round(float(i),2) for i in temp]

maxAmtPerCustomer[key]=max(temp)

print("CustomerId\t\t\tMinAmtperCustomer")

maxAmtPerCustomer\_sorted = OrderedDict(sorted(maxAmtPerCustomer.items(), key=lambda x: float(x[1])))

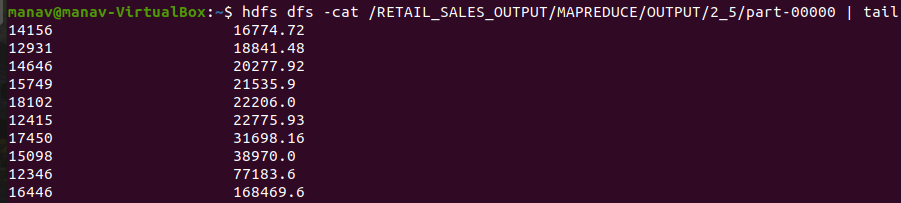
for (key,value) in maxAmtPerCustomer\_sorted.items():

print(key,'\t\t\t',str(value))

##### Execution Command:

hadoop jar hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/MaxAmtperCustomer\_mapper2\_5.py -mapper "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/MaxAmtperCustomer\_mapper2\_5.py" -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/MaxAmtperCustomer\_reducer2\_5.py -reducer "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/MaxAmtperCustomer\_reducer2\_5.py" -input /RETAIL\_SALES/OnlineRetail.txt -output /RETAIL\_SALES\_OUTPUT/MAPREDUCE/OUTPUT/2\_5

##### Output:



#### AvgAmtperCustomer:

##### AvgAmtperCustomer\_mapper2\_6.py

#!usr/bin/python3

import sys

for row in sys.stdin:

col = row.strip().split('\t')

try:

invoiceNum, unitprice, quantity, custID = col[0], float(col[5]), int(col[3]), col[6]

if( unitprice >=0 and quantity >0 and str(custID) != ''):

print('%s\t%s\t%s' % (invoiceNum, custID, unitprice\*quantity))

except ValueError: pass

##### AvgAmtperCustomer\_reucer2\_6.py

#!usr/bin/python3

import sys, math

from collections import OrderedDict

from itertools import groupby

revenueAggregate = {}

groupRevenue = {}

avgAmtPerCustomer = {}

for row in sys.stdin:

invoiceNum, custId, UQ = row.strip().split('\t',2)

try:

revenueAggregate[custId+'\_'+invoiceNum] = revenueAggregate[custId+'\_'+invoiceNum] + float(UQ)

except:

revenueAggregate[custId+'\_'+invoiceNum] = float(UQ)

revenueSorted = OrderedDict(sorted(revenueAggregate.items(), key=lambda x: float(x[0])))

for (key,value) in revenueSorted.items():

custId, invoiceNum = key.split('\_',1)

try:

groupRevenue[custId] = groupRevenue[custId] + ',' +str(value)

except:

groupRevenue[custId] = str(value)

for (key,value) in groupRevenue.items():

temp = value.split(',')

temp = [round(float(i),2) for i in temp]

avgAmtPerCustomer[key]=round((sum(temp)/len(temp)),2)

print("CustomerId\t\t\tAvgAmtperCustomer")

SortedAvgAmtPerCustomer = OrderedDict(sorted(avgAmtPerCustomer.items(), key=lambda x: float(x[1])))

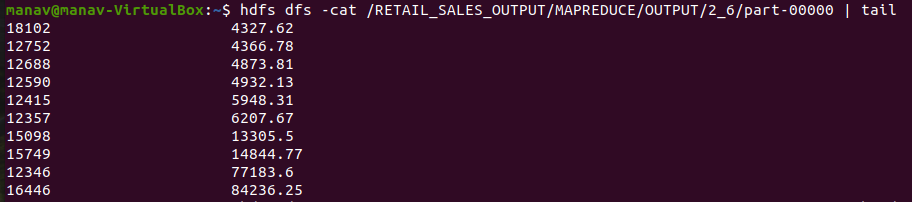
for (key,value) in SortedAvgAmtPerCustomer.items():

print(key,'\t\t\t',str(value))

##### Execution Command:

hadoop jar hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/AvgAmtperCustomer\_mapper2\_6.py -mapper "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/AvgAmtperCustomer\_mapper2\_6.py" -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/AvgAmtperCustomer\_reducer2\_6.py -reducer "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/AvgAmtperCustomer\_reducer2\_6.py" -input /RETAIL\_SALES/OnlineRetail.txt -output /RETAIL\_SALES\_OUTPUT/MAPREDUCE/OUTPUT/2\_6

##### Output:



### Spark

val OnlineRetail = sc.textFile(InputFile).map(row => row.split("\t"))

#### NumCustomers:

##### Spark Commands:

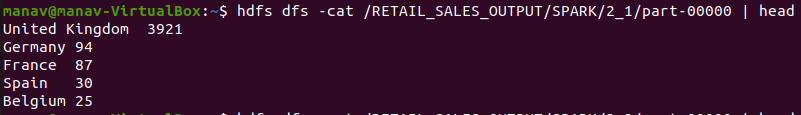
val FilterMap = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(COUNTRY).toString, col(CUSTOMERID).toString)})

val CustIdCount = FilterMap.distinct().groupByKey().mapValues(sq => (sq.size)).sortBy(\_.\_2,false)

val NumCustomers = sc.parallelize(CustIdCount.take(5))

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

##### Output:



#### NumTransactions:

##### Spark Commands:

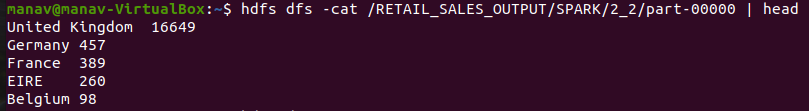
val FilterMap = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(COUNTRY).toString, col(INVOICENO).toString)})

val TransactionCount = FilterMap.distinct().groupByKey().mapValues(sq => (sq.size)).sortBy(\_.\_2,false)

val NumTransactions = sc.parallelize(TransactionCount.take(5))

NumTransactions.map(r => r.productIterator.mkString("\t")).saveAsTextFile(OutputFile+"2\_2")

##### Output:



#### AvgNumItems:

##### Spark Commands:

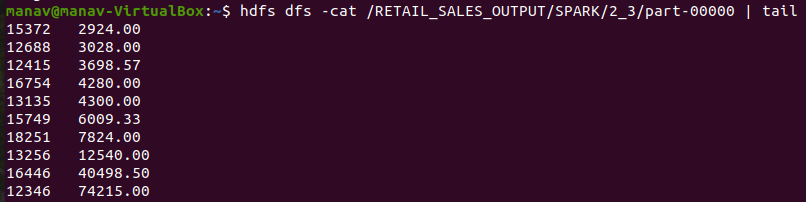
val FilterMap = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(INVOICENO).toString + "\_" + col(CUSTOMERID).toString, col(QUANTITY).toInt)})

val SumUQ = FilterMap.groupByKey().mapValues(sq => (sq.sum)).map(col => {(col.\_1.toString.split("\_")(1),col.\_2.toDouble)})

val AvgNumItems = SumUQ.groupByKey().mapValues(sq => (BigDecimal(sq.sum/sq.size).setScale(2, BigDecimal.RoundingMode.HALF\_UP))).sortBy(\_.\_2)

AvgNumItems.map(r => r.productIterator.mkString("\t")).saveAsTextFile(OutputFile+"2\_3")

##### Output:



#### MinAmtperCustomer:

##### Spark Commands:

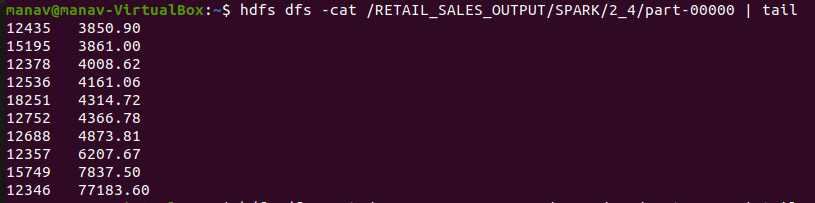
val FilterMap = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(INVOICENO).toString + "\_" + col(CUSTOMERID).toString, col(QUANTITY).toInt \* BigDecimal(col(UNITPRICE)).setScale(2, BigDecimal.RoundingMode.HALF\_UP))})

val SumUQ = FilterMap.groupByKey().mapValues(sq => (sq.sum)).map(col => {(col.\_1.toString.split("\_")(1),col.\_2)})

val MinAmtperCustomer = SumUQ.groupByKey().mapValues(sq => (sq.min)).sortBy(\_.\_2)

MinAmtperCustomer.map(r => r.productIterator.mkString("\t")).saveAsTextFile(OutputFile+"2\_4")

##### Output:



#### MaxAmtperCustomer:

##### Spark Commands:

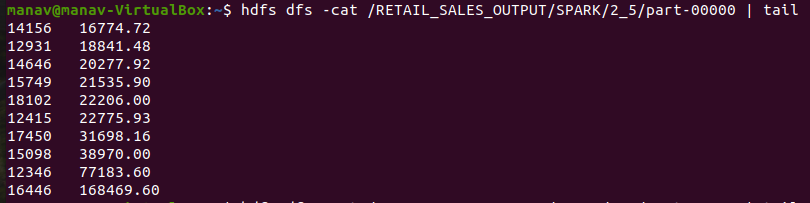
val FilterMap = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(INVOICENO).toString + "\_" + col(CUSTOMERID).toString, col(QUANTITY).toInt \* BigDecimal(col(UNITPRICE)).setScale(2, BigDecimal.RoundingMode.HALF\_UP))})

val SumUQ = FilterMap.groupByKey().mapValues(sq => (sq.sum)).map(col => {(col.\_1.toString.split("\_")(1),col.\_2)})

val MaxAmtperCustomer = SumUQ.groupByKey().mapValues(sq => (sq.max)).sortBy(\_.\_2)

MaxAmtperCustomer.map(r => r.productIterator.mkString("\t")).saveAsTextFile(OutputFile+"2\_5")

##### Output:



#### AvgAmtperCustomer

##### Spark Commands:

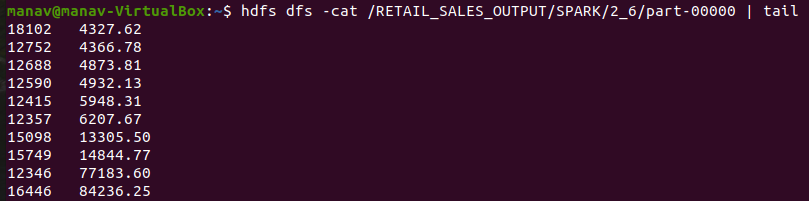
val FilterMap = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(INVOICENO).toString + "\_" + col(CUSTOMERID).toString, col(QUANTITY).toInt \* BigDecimal(col(UNITPRICE)).setScale(2, BigDecimal.RoundingMode.HALF\_UP))})

val SumUQ = FilterMap.groupByKey().mapValues(sq => (sq.sum)).map(col => {(col.\_1.toString.split("\_")(1),col.\_2.toDouble)})

val AvgAmtperCustomer = SumUQ.groupByKey().mapValues(sq => (BigDecimal(sq.sum/sq.size).setScale(2, BigDecimal.RoundingMode.HALF\_UP))).sortBy(\_.\_2)

AvgAmtperCustomer.map(r => r.productIterator.mkString("\t")).saveAsTextFile(OutputFile+"2\_6")

##### Output:



## Question 3

### 3.Daily Sales Activity like NumVisits and TotalAmt monthly and quarterly for each year:

### Calculate total number of customers visted and total amount of money spent by customers per month and per quarter.

## Solution 3:

### Hive

#### Monthly

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/3\_1'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT SUBSTR(InvoiceDate,1,7) AS YEAR\_MONTH, COUNT(CUSTOMERID) AS NumVisits, ROUND(SUM(UNITPRICE \* QUANTITY),2) AS TotalAmt

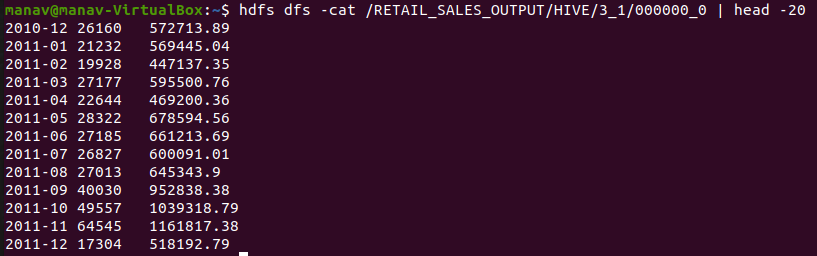
FROM UC1\_RETAIL.ONLINERETAIL

WHERE UNITPRICE >=0 and QUANTITY >0 and TRIM(CUSTOMERID) <> ''

GROUP BY SUBSTR(InvoiceDate,1,7)

ORDER BY YEAR\_MONTH;

##### Output:



#### Quarterly

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/3\_2'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT CONCAT(SUBSTR(InvoiceDate, 1, 5), 'Q', QUARTER(SUBSTR(InvoiceDate,1,10))) AS QUARTER, COUNT(CUSTOMERID) AS NumVisits, ROUND(SUM(UNITPRICE \* QUANTITY),2) AS TotalAmt

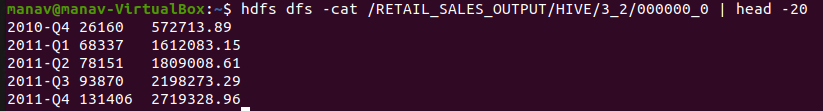
FROM UC1\_RETAIL.ONLINERETAIL

WHERE UNITPRICE >=0 and QUANTITY >0 and TRIM(CUSTOMERID) <> ''

GROUP BY CONCAT(SUBSTR(InvoiceDate,1,5), 'Q', QUARTER(SUBSTR(InvoiceDate,1,10)))

ORDER BY QUARTER;

##### Output:



### Pig

#### Monthly

##### Pig Commands:

RETAIL = LOAD 'hdfs://localhost:9000/RETAIL\_SALES/' USING PigStorage('\t') AS ( INVOICENO:INT, STOCKCODE:CHARARRAY, DESCRIPTION:CHARARRAY, QUANTITY:INT, INVOICEDATE:CHARARRAY, UNITPRICE:DOUBLE, CUSTOMERID:CHARARRAY, COUNTRY:CHARARRAY );

FILTER\_DATA = FILTER RETAIL BY UNITPRICE >=0 AND QUANTITY >0 AND CUSTOMERID != '';

MONTH\_GROUP = GROUP FILTER\_DATA BY SUBSTRING(INVOICEDATE,0,7);

NUMVISITS\_TOTALAMT\_M = FOREACH MONTH\_GROUP {

TOTAL\_PQ = FOREACH FILTER\_DATA GENERATE UNITPRICE \* QUANTITY AS (PQ:FLOAT);

GENERATE group, COUNT(FILTER\_DATA.CUSTOMERID) AS CUSTID\_COUNT, ROUND\_TO(SUM(TOTAL\_PQ.PQ),2) AS (REVENUE\_AGGREGATE:FLOAT);

};

STORE NUMVISITS\_TOTALAMT\_M INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/3\_1/' USING PigStorage ('\t');

##### G:\PGD_class\Retail-Sales-Big-Data\OUTPUT\pig_3_1.PNGOutput:

#### Quarterly

##### Pig Commands:

FILTER\_DATA = FILTER RETAIL BY UNITPRICE >=0 AND QUANTITY >0 AND CUSTOMERID != '';

QUARTER\_GROUP = GROUP FILTER\_DATA BY CONCAT(SUBSTRING(INVOICEDATE,0,5), 'Q', (CHARARRAY)(INT)(FLOOR(((INT)SUBSTRING(INVOICEDATE, 5,7)- 1) / 3) + 1));

NUMVISITS\_TOTALAMT\_Q = FOREACH QUARTER\_GROUP {

TOTAL\_PQ = FOREACH FILTER\_DATA GENERATE UNITPRICE \* QUANTITY AS (PQ:FLOAT);

GENERATE group, COUNT(FILTER\_DATA.CUSTOMERID) AS CUSTID\_COUNT, ROUND\_TO(SUM(TOTAL\_PQ.PQ),2) AS (REVENUE\_AGGREGATE:FLOAT);

};

STORE NUMVISITS\_TOTALAMT\_Q INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/3\_2/' USING PigStorage ('\t');

##### G:\PGD_class\Retail-Sales-Big-Data\OUTPUT\pig_3_2.PNGOutput:

### MapReduce

##### NumVisits\_TotalAmt\_MQ\_mapper3.py

#!usr/bin/python3

import sys, math

for row in sys.stdin:

col = row.strip().split('\t')

try:

invoiceDate, unitprice, quantity, custID = col[4], float(col[5]), int(col[3]), col[6].strip()

yyyymm = invoiceDate[0:7]

q = invoiceDate[0:5] + 'Q' + str(int(math.floor((int(invoiceDate[5:7])- 1) / 3) + 1))

if( unitprice >=0 and quantity >0 and str(custID) != ''):

print('%s\t%s\t%s\t%s' % (yyyymm, q, custID, unitprice\*quantity))

except ValueError: pass

##### NumVisits\_TotalAmt\_MQ\_reducer3.py

#!usr/bin/python3

import sys

from collections import OrderedDict

MothlyRevenue = {}

MothlyNumVisits = {}

QuarterlyRevenue = {}

QuarterlyNumVisits = {}

input=[]

for row in sys.stdin:

input.append(row)

mm, q, custId, UQ = row.strip().split('\t',3)

UQ = float(UQ)

try:

MothlyRevenue[mm] = MothlyRevenue[mm] + UQ

MothlyNumVisits[mm] = MothlyNumVisits[mm] + 1

except:

MothlyRevenue[mm] = UQ

MothlyNumVisits[mm] = 1

for row in input:

mm, q, custId, UQ = row.strip().split('\t',3)

try:

UQ = float(UQ)

except ValueError: pass

try:

QuarterlyRevenue[q] = QuarterlyRevenue[q] + UQ

QuarterlyNumVisits[q] = QuarterlyNumVisits[q] + 1

except:

QuarterlyRevenue[q] = UQ

QuarterlyNumVisits[q] = 1

SortedMothlyRevenue = OrderedDict(sorted(MothlyRevenue.items(), key=lambda x: x[0]))

SortedMothlyNumVisits = OrderedDict(sorted(MothlyNumVisits.items(), key=lambda x: x[0]))

SortedQuarterlyRevenue = OrderedDict(sorted(QuarterlyRevenue.items(), key=lambda x: x[0]))

SortedQuarterlyNumVisits = OrderedDict(sorted(QuarterlyNumVisits.items(), key=lambda x: x[0]))

print("MONTH\t\t\tNumVisits\t\t\tTotalAmt")

for (key,value), (key2,value2) in zip(SortedMothlyRevenue.items(), SortedMothlyNumVisits.items()):

print(key, "\t\t\t", value2, "\t\t\t", str(round(value,2)))

print("\n\nQUARTER\t\t\tNumVisits\t\t\tTotalAmt")

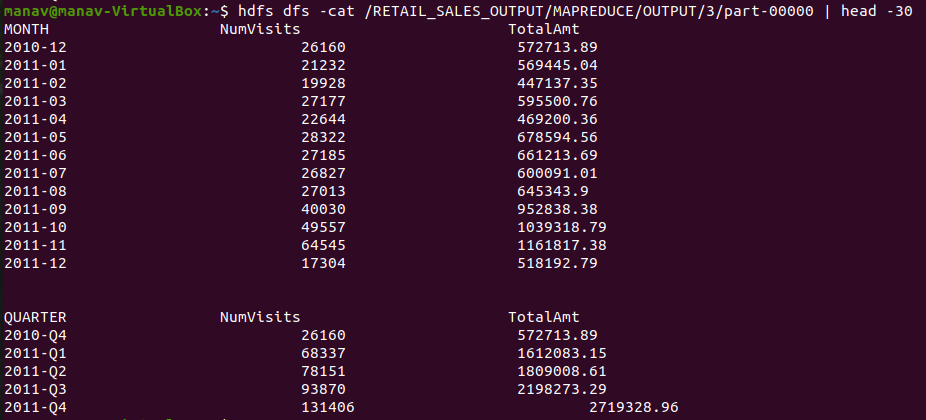
for (key,value), (key2,value2) in zip(SortedQuarterlyRevenue.items(), SortedQuarterlyNumVisits.items()):

print(key, "\t\t\t", value2, "\t\t\t", str(round(value,2)))

##### Execution Command:

hadoop jar hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumVisits\_TotalAmt\_MQ\_mapper3.py -mapper "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumVisits\_TotalAmt\_MQ\_mapper3.py" -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumVisits\_TotalAmt\_MQ\_reducer3.py -reducer "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumVisits\_TotalAmt\_MQ\_reducer3.py" -input /RETAIL\_SALES/OnlineRetail.txt -output /RETAIL\_SALES\_OUTPUT/MAPREDUCE/OUTPUT/3

##### Output:



### Spark

#### MONTHLY:

##### Spark Commands:

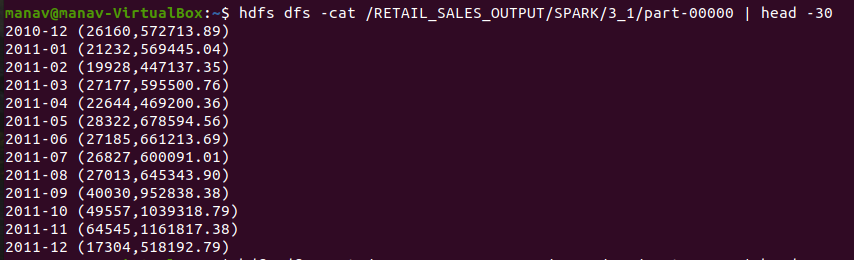
val OnlineRetail = sc.textFile(InputFile).map(row => row.split("\t"))

val FilterMap = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(INVOICEDATE).toString.split("-")(0)+"-"+col(INVOICEDATE).toString.split("-")(1), col(QUANTITY).toInt \* BigDecimal(col(UNITPRICE)).setScale(2, BigDecimal.RoundingMode.HALF\_UP))})

val NumVisits\_TotalAmt\_M = FilterMap.groupByKey().mapValues(sq => (sq.size,sq.sum)).sortBy(\_.\_1)

NumVisits\_TotalAmt\_M.map(r => r.productIterator.mkString("\t")).saveAsTextFile(OutputFile+"3\_1")

##### Output:



#### QUARTERLY:

##### Spark Commands:

val OnlineRetail = sc.textFile(InputFile).map(row => row.split("\t"))

val FilterMap = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(INVOICEDATE).toString.split("-")(0)+"-Q"+(((col(INVOICEDATE).toString.split("-")(1).toInt-1)/3).floor+1).toInt, col(QUANTITY).toInt \* BigDecimal(col(UNITPRICE)).setScale(2, BigDecimal.RoundingMode.HALF\_UP))})

val NumVisits\_TotalAmt\_Q = FilterMap.groupByKey().mapValues(sq => (sq.size,sq.sum)).sortBy(\_.\_1)

NumVisits\_TotalAmt\_Q.map(r => r.productIterator.mkString("\t")).saveAsTextFile(OutputFile+"3\_2")

##### G:\PGD_class\Retail-Sales-Big-Data\OUTPUT\spark_3_2.PNGOutput:

## Question 4

### 4. Hourly sales Activity like NumVisits and TotalAmt per hour of day:

### Calculate total number of customers visted and total amount of money spent by customers per hour.

## Solution 4:

### Hive

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/4'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT CAST(SUBSTR(INVOICEDATE, 12, INSTR(INVOICEDATE,':')-12) AS INT) AS HH, COUNT(CUSTOMERID) AS NUMVISITS, ROUND(SUM(UNITPRICE \* QUANTITY),2) AS TOTALAMT

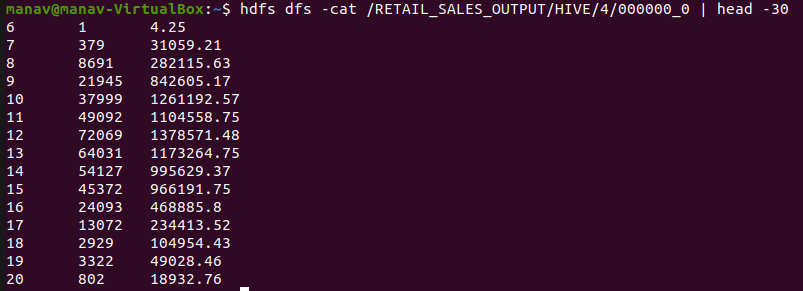
FROM UC1\_RETAIL.ONLINERETAIL

WHERE UNITPRICE >=0 AND QUANTITY >0 AND TRIM(CUSTOMERID) <> ''

GROUP BY CAST(SUBSTR(INVOICEDATE, 12, INSTR(INVOICEDATE,':')-12) AS INT)

ORDER BY HH;

##### Output:



### Pig

##### Pig Commands:

RETAIL = LOAD 'hdfs://localhost:9000/RETAIL\_SALES/' USING PigStorage('\t') AS ( INVOICENO:INT, STOCKCODE:CHARARRAY, DESCRIPTION:CHARARRAY, QUANTITY:INT, INVOICEDATE:CHARARRAY, UNITPRICE:DOUBLE, CUSTOMERID:CHARARRAY, COUNTRY:CHARARRAY );

FILTER\_DATA = FILTER RETAIL BY UNITPRICE >=0 AND QUANTITY >0 AND CUSTOMERID != '';

HOUR\_GROUP = GROUP FILTER\_DATA BY SUBSTRING(INVOICEDATE,11,INDEXOF(INVOICEDATE,':'));

NUMVISITS\_TOTALAMT\_H = FOREACH HOUR\_GROUP {

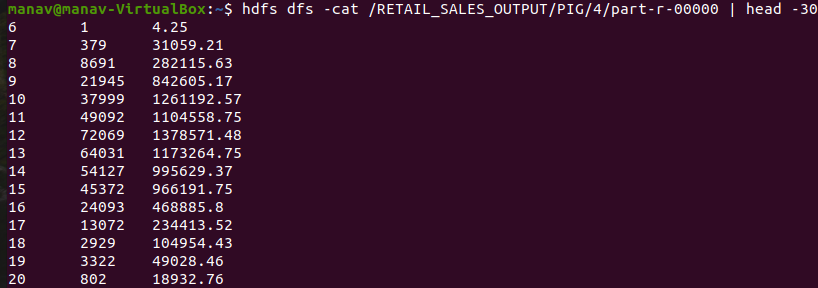
TOTAL\_PQ = FOREACH FILTER\_DATA GENERATE UNITPRICE \* QUANTITY AS (PQ:FLOAT);

GENERATE group, COUNT(FILTER\_DATA.CUSTOMERID) AS CUSTID\_COUNT, ROUND\_TO(SUM(TOTAL\_PQ.PQ),2) AS (REVENUE\_AGGREGATE:FLOAT);

};

STORE NUMVISITS\_TOTALAMT\_H INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/4/' USING PigStorage ('\t');

##### Output:



### MapReduce

##### NumVisits\_TotalAmt\_H\_mapper4.py

#!usr/bin/python3

import sys, math

for row in sys.stdin:

col = row.strip().split('\t')

try:

invoiceDate, unitprice, quantity, custID = col[4], float(col[5]), int(col[3]), col[6].strip()

hh = invoiceDate[11:invoiceDate.find(":")]

if( unitprice >=0 and quantity >0 and str(custID) != ''):

print('%s\t%s\t%s' % (hh, custID, unitprice\*quantity))

except ValueError: pass

##### NumVisits\_TotalAmt\_H\_reducer4.py

#!usr/bin/python3

import sys

from collections import OrderedDict

HourlyRevenue = {}

HourlyNumVisits = {}

for row in sys.stdin:

hh, custId, UQ = row.strip().split('\t',2)

UQ = float(UQ)

try:

HourlyRevenue[hh] = HourlyRevenue[hh] + UQ

HourlyNumVisits[hh] = HourlyNumVisits[hh] + 1

except:

HourlyRevenue[hh] = UQ

HourlyNumVisits[hh] = 1

SortedHourlyRevenue = OrderedDict(sorted(HourlyRevenue.items(), key=lambda x: int(x[0])))

SortedHourlyNumVisits = OrderedDict(sorted(HourlyNumVisits.items(), key=lambda x: int(x[0])))

print("Hour\t\t\tNumVisits\t\t\tTotalAmt")

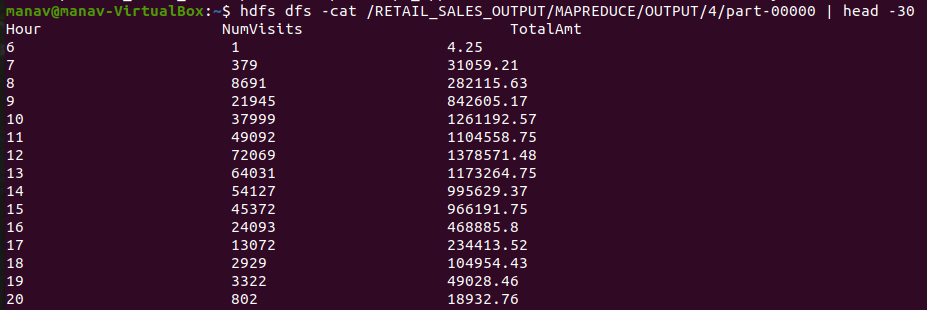
for (key,value), (key2,value2) in zip(SortedHourlyRevenue.items(), SortedHourlyNumVisits.items()):

print(key, "\t\t\t", value2, "\t\t\t", str(round(value,2)))

##### Execution Command:

hadoop jar hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumVisits\_TotalAmt\_H\_mapper4.py -mapper "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumVisits\_TotalAmt\_H\_mapper4.py" -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumVisits\_TotalAmt\_H\_reducer4.py -reducer "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/NumVisits\_TotalAmt\_H\_reducer4.py" -input /RETAIL\_SALES/OnlineRetail.txt -output /RETAIL\_SALES\_OUTPUT/MAPREDUCE/OUTPUT/4

##### Output:



### Spark

##### Spark Commands:

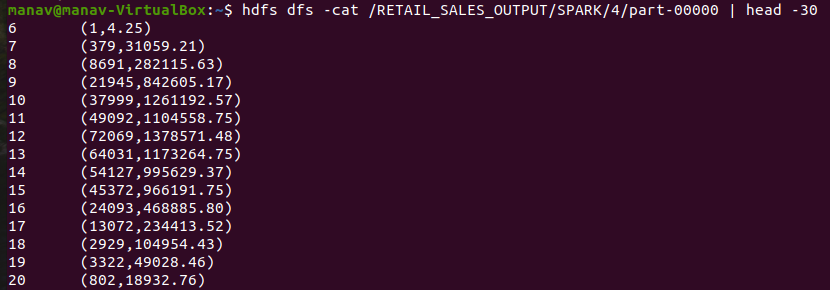
val OnlineRetail = sc.textFile(InputFile).map(row => row.split("\t"))

val FilterMap = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(INVOICEDATE).toString.split(" ")(1).split(":")(0).toInt, col(QUANTITY).toInt \* BigDecimal(col(UNITPRICE)).setScale(2, BigDecimal.RoundingMode.HALF\_UP))})

val NumVisits\_TotalAmt\_H = FilterMap.groupByKey().mapValues(sq => (sq.size,sq.sum)).sortBy(\_.\_1)

NumVisits\_TotalAmt\_H.map(r => r.productIterator.mkString("\t")).saveAsTextFile(OutputFile+"4")

##### Output:



## Question 5

### 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.):

### Calculate number of items purchased per each transaction and group transaction with same number of items and count its frequency.

## Solution 5:

### Hive

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/5'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT SUM\_Q, COUNT(SUM\_Q) AS BASKET\_SIZE\_COUNT FROM (

SELECT INVOICENO, SUM(QUANTITY) AS SUM\_Q

FROM UC1\_RETAIL.ONLINERETAIL

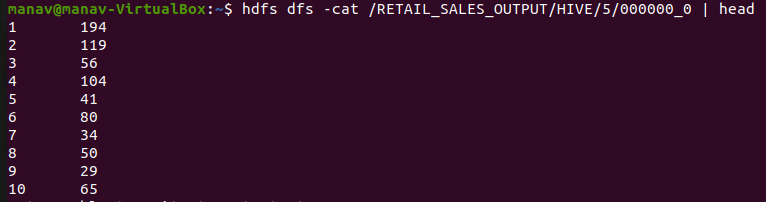
WHERE UNITPRICE >=0 AND QUANTITY >0 AND TRIM(CUSTOMERID) <> ''

GROUP BY INVOICENO ) RETAIL\_TEMP

GROUP BY SUM\_Q

ORDER BY SUM\_Q;

##### Output:



### Pig

##### Pig Commands:

RETAIL = LOAD 'hdfs://localhost:9000/RETAIL\_SALES/' USING PigStorage('\t') AS ( INVOICENO:INT, STOCKCODE:CHARARRAY, DESCRIPTION:CHARARRAY, QUANTITY:INT, INVOICEDATE:CHARARRAY, UNITPRICE:DOUBLE, CUSTOMERID:CHARARRAY, COUNTRY:CHARARRAY );

FILTER\_DATA = FILTER RETAIL BY UNITPRICE >=0 AND QUANTITY >0 AND CUSTOMERID != '';

INVOICE\_GROUP = GROUP FILTER\_DATA BY INVOICENO;

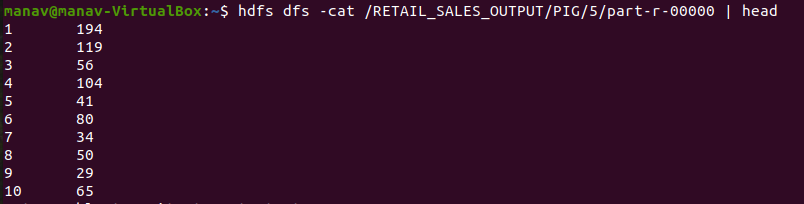
SUM\_QUANTITY = FOREACH INVOICE\_GROUP GENERATE group, SUM(FILTER\_DATA.QUANTITY) AS (BASKET\_SUMQ:LONG);

BASKET = GROUP SUM\_QUANTITY BY BASKET\_SUMQ;

COUNT\_BASKET= FOREACH BASKET GENERATE group, COUNT(SUM\_QUANTITY.BASKET\_SUMQ) AS (COUNT\_BASKET\_SUMQ:LONG);

SORT\_BASKET\_SUMQ = ORDER COUNT\_BASKET BY group;

STORE SORT\_BASKET\_SUMQ INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/5/' USING PigStorage ('\t');



##### Output:

### MapReduce

##### BasketSize\_mapper5.py

#!usr/bin/python3

import sys, math

for row in sys.stdin:

col = row.strip().split('\t')

try:

invoiceNum, unitprice, quantity, custID = col[0].strip(), float(col[5]), int(col[3]), col[6].strip()

if( unitprice >=0 and quantity >0 and str(custID) != ''):

print('%s\t%s' % (invoiceNum, quantity))

except ValueError: pass

##### BasketSize\_reducer5.py

#!usr/bin/python3

import sys

from collections import OrderedDict

quantityCount = {}

BasketSize = {}

for row in sys.stdin:

invoiceNum, qty = row.strip().split('\t',1)

try:

qty = int(qty)

except ValueError: pass

try:

quantityCount[invoiceNum] = quantityCount[invoiceNum] + qty

except:

quantityCount[invoiceNum] = qty

SortedQuantityCount = OrderedDict(sorted(quantityCount.items(), key=lambda x: int(x[1])))

for (key,value) in SortedQuantityCount.items():

try:

BasketSize[value] = BasketSize[value] + 1

except:

BasketSize[value] = 1

SortedBasketSize = OrderedDict(sorted(BasketSize.items(), key=lambda x: int(x[0])))

print("Basket Size\t\t\tCount")

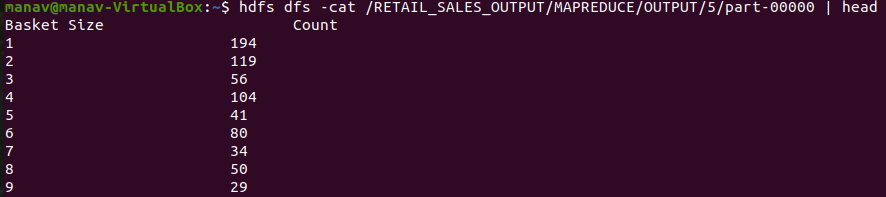
for (key,value) in SortedBasketSize.items():

print(key, "\t\t\t", value)

##### Execution Command:

hadoop jar hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/BasketSize\_mapper5.py -mapper "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/BasketSize\_mapper5.py" -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/BasketSize\_reducer5.py -reducer "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/BasketSize\_reducer5.py" -input /RETAIL\_SALES/OnlineRetail.txt -output /RETAIL\_SALES\_OUTPUT/MAPREDUCE/OUTPUT/5

##### Output:



### Spark

##### Spark Commands:

val OnlineRetail = sc.textFile(InputFile).map(row => row.split("\t"))

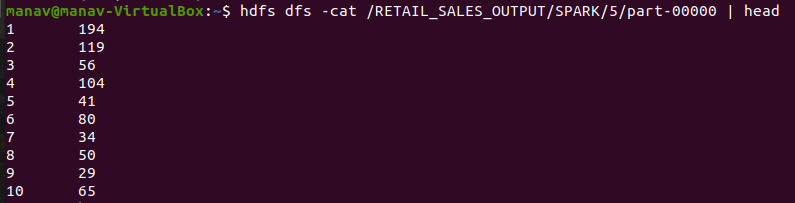
val FilterMap = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(INVOICENO).toString, col(QUANTITY).toInt)})

val SumQ = FilterMap.groupByKey().mapValues(sq => (sq.sum)).map(col => {(col.\_2,col.\_1)})

val BasketSizeCount = SumQ.groupByKey().mapValues(sq => (sq.size)).sortBy(\_.\_1)

BasketSizeCount.map(r => r.productIterator.mkString("\t")).saveAsTextFile(OutputFile+"5")

##### Output:



## Question 6

### 6. Top 20 Items sold by frequency:

### Count number of times each item is sold and display top 20 items that are sold most.

## Solution 6:

### Hive

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/6'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT STOCKCODE, SUM(QUANTITY) AS SUM\_Q

FROM UC1\_RETAIL.ONLINERETAIL

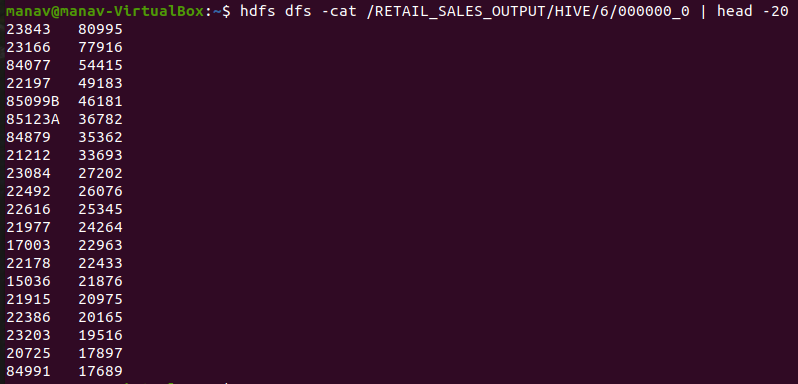
WHERE UNITPRICE >=0 AND QUANTITY >0 AND TRIM(CUSTOMERID) <> ''

GROUP BY STOCKCODE

ORDER BY SUM\_Q DESC

LIMIT 20;

##### Output:



### Pig

##### Pig Commands:

RETAIL = LOAD 'hdfs://localhost:9000/RETAIL\_SALES/' USING PigStorage('\t') AS ( INVOICENO:INT, STOCKCODE:CHARARRAY, DESCRIPTION:CHARARRAY, QUANTITY:INT, INVOICEDATE:CHARARRAY, UNITPRICE:DOUBLE, CUSTOMERID:CHARARRAY, COUNTRY:CHARARRAY );

FILTER\_DATA = FILTER RETAIL BY UNITPRICE >=0 AND QUANTITY >0 AND CUSTOMERID != '';

STOCK\_GROUP = GROUP FILTER\_DATA BY STOCKCODE;

SUM\_Q = FOREACH STOCK\_GROUP GENERATE group, SUM(FILTER\_DATA.QUANTITY) AS (FREQUENCY\_STOCK:LONG);

SORT\_FREQUENCY\_STOCK = ORDER SUM\_Q BY FREQUENCY\_STOCK DESC;

STOCK\_FREQUENCY = LIMIT SORT\_FREQUENCY\_STOCK 20;

STORE STOCK\_FREQUENCY INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/6/' USING PigStorage ('\t');

##### G:\PGD_class\Retail-Sales-Big-Data\OUTPUT\pig_6.PNGOutput:

### MapReduce

##### ItemFrequency mapper6.py

#!usr/bin/python3

import sys, math

for row in sys.stdin:

col = row.strip().split('\t')

try:

stockCode, unitprice, quantity, custID = col[1], float(col[5]), int(col[3]), col[6].strip()

if( unitprice >=0 and quantity >0 and str(custID) != ''):

print('%s\t%s' % (stockCode, quantity))

except ValueError: pass

##### ItemFrequency.py

#!usr/bin/python3

import sys

from collections import OrderedDict

quantitySum = {}

for row in sys.stdin:

stockCode, quantity = row.strip().split('\t',1)

quantity = int(quantity)

try:

quantitySum[stockCode] = quantitySum[stockCode] + quantity

except:

quantitySum[stockCode] = quantity

SortedItemFrequency = OrderedDict(sorted(quantitySum.items(), key=lambda x: int(x[1]), reverse=True))

print("Stock Code\t\t\tCount")

c=0

for (key,value) in SortedItemFrequency.items():

c=c+1

if(c<=20):

print(key, "\t\t\t", value)

##### Execution Command:

hadoop jar hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/ItemFrequency\_mapper6.py -mapper "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/ItemFrequency\_mapper6.py" -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/ItemFrequency\_reducer6.py -reducer "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/ItemFrequency\_reducer6.py" -input /RETAIL\_SALES/OnlineRetail.txt -output /RETAIL\_SALES\_OUTPUT/MAPREDUCE/OUTPUT/6

##### G:\PGD_class\Retail-Sales-Big-Data\OUTPUT\mapreduce_6.PNGOutput:

### Spark

##### Spark Commands:

val OnlineRetail = sc.textFile(InputFile).map(row => row.split("\t"))

val stockcode\_filter = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(STOCKCODE).toString, col(QUANTITY).toInt)})

val SumQ = stockcode\_filter.groupByKey().mapValues(sq => (sq.sum)).sortBy(\_.\_2,false)

val ItemsFrequency = sc.parallelize(SumQ.take(20))

ItemsFrequency.map(r => r.productIterator.mkString("\t")).saveAsTextFile(OutputFile+"6")

##### Output:



## Question 7

### 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:

### Calculate CLV by adding all total amount spend by each customer and group each CLV in the interval’s of 1000’s.

## Solution 7:

### Hive

##### Hive Commands:

INSERT OVERWRITE DIRECTORY '/RETAIL\_SALES\_OUTPUT/HIVE/7'

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t'

STORED AS TEXTFILE

SELECT SUM\_UQ, (SUM\_UQ\*1000 ||'-'|| (SUM\_UQ+1)\*1000) AS CLV\_INTERVAL, COUNT(CUSTOMERID) AS COUNT\_CUSTOMER FROM (

SELECT CEIL(SUM(UNITPRICE \* QUANTITY)/1000) AS SUM\_UQ, CUSTOMERID

FROM UC1\_RETAIL.ONLINERETAIL

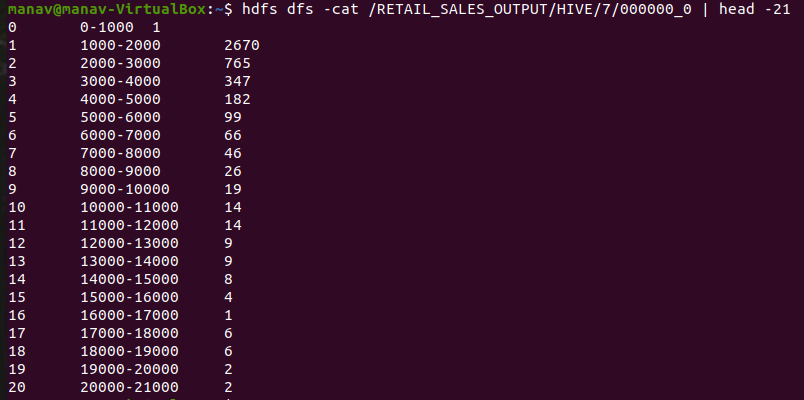
WHERE UNITPRICE >=0 AND QUANTITY >0 AND TRIM(CUSTOMERID) <> ''

GROUP BY CUSTOMERID ) RETAIL\_TEMP

GROUP BY SUM\_UQ

ORDER BY SUM\_UQ;

##### Output:



### Pig

##### Pig Commands:

RETAIL = LOAD 'hdfs://localhost:9000/RETAIL\_SALES/' USING PigStorage('\t') AS ( INVOICENO:INT, STOCKCODE:CHARARRAY, DESCRIPTION:CHARARRAY, QUANTITY:INT, INVOICEDATE:CHARARRAY, UNITPRICE:DOUBLE, CUSTOMERID:CHARARRAY, COUNTRY:CHARARRAY );

FILTER\_DATA = FILTER RETAIL BY UNITPRICE >=0 AND QUANTITY >0 AND CUSTOMERID != '';

CUSTID\_GROUP = GROUP FILTER\_DATA BY CUSTOMERID;

REVENUE = FOREACH CUSTID\_GROUP {

TOTAL\_PQ = FOREACH FILTER\_DATA GENERATE UNITPRICE \* QUANTITY AS (PQ:DOUBLE);

GENERATE group, (LONG)CEIL(SUM(TOTAL\_PQ.PQ)/1000) AS (CUST\_LIFETIME\_VALUE:LONG);

};

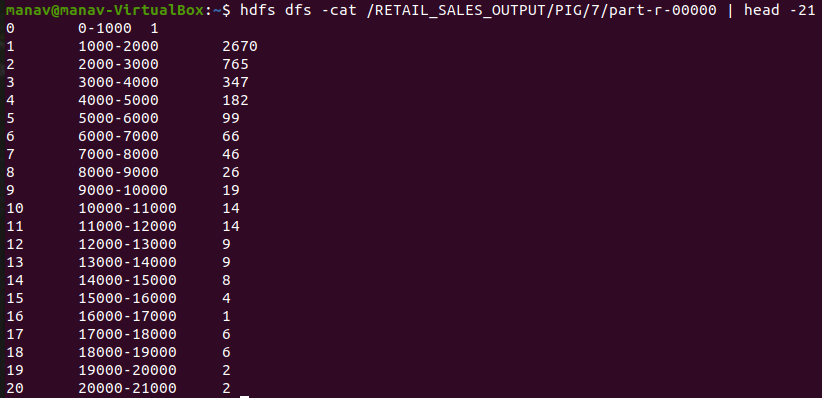
CLV\_GROUP= GROUP REVENUE BY CUST\_LIFETIME\_VALUE;

CLV\_INTERVAL= FOREACH CLV\_GROUP GENERATE group, CONCAT((CHARARRAY)(group\*1000),'-',(CHARARRAY)((group+1)\*1000)) AS CUST\_LIFETIME\_VALUE\_RANGE, COUNT(REVENUE.group) AS (COUNT\_CUSTOMER:LONG);

SORT\_CUST\_LIFETIME\_VAL = ORDER CLV\_INTERVAL BY group;

STORE SORT\_CUST\_LIFETIME\_VAL INTO 'hdfs://localhost:9000/RETAIL\_SALES\_OUTPUT/PIG/7/' USING PigStorage ('\t');

##### Output:



### MapReduce

##### CLV\_mapper7.py

#!usr/bin/python3

import sys, math

for row in sys.stdin:

col = row.strip().split('\t')

try:

unitprice, quantity, custID = float(col[5]), int(col[3]), col[6].strip()

if( unitprice >=0 and quantity >0 and str(custID) != ''):

print('%s\t%s' % (custID, unitprice\*quantity))

except ValueError: pass

##### CLV\_reducer7.py

#!usr/bin/python3

import sys, math

from collections import OrderedDict

revenueAggregate = {}

SortedRevenue = {}

CLV = {}

SortedCLV = {}

for row in sys.stdin:

custId, UQ = row.strip().split('\t', 1)

try:

UQ = float(UQ)

except ValueError: pass

try:

revenueAggregate[custId] = revenueAggregate[custId] + UQ

except:

revenueAggregate[custId] = UQ

for (key,value) in revenueAggregate.items():

value=math.ceil(float(value)/1000)

SortedRevenue[key]=value

for (key,value) in SortedRevenue.items():

try:

CLV[value] = CLV[value] + 1

except:

CLV[value] = 1

SortedCLV = OrderedDict(sorted(CLV.items(), key=lambda x: int(x[0])))

print("CLV\t\t\tCLV RANGE\t\t\tCUSTOMERID\_COUNT")

for (key,value) in SortedCLV.items():

print(key, "\t\t\t", str(key\*1000), '-', str((key+1)\*1000), "\t\t\t", str(value))

##### Execution Command:

hadoop jar hadoop-3.1.2/share/hadoop/tools/lib/hadoop-streaming-3.1.2.jar -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/CLV\_mapper7.py -mapper "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/CLV\_mapper7.py" -file /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/CLV\_reducer7.py -reducer "python3 /home/manav/Documents/RETAIL\_SALES\_ASSIGNMENT/MapReduce/CLV\_reducer7.py" -input /RETAIL\_SALES/OnlineRetail.txt -output /RETAIL\_SALES\_OUTPUT/MAPREDUCE/OUTPUT/7

##### G:\PGD_class\Retail-Sales-Big-Data\OUTPUT\mapreduce_7.PNGOutput:

### Spark

##### Spark Commands:

val OnlineRetail = sc.textFile(InputFile).map(row => row.split("\t"))

val FilterMap = OnlineRetail.filter(col => col(QUANTITY).toInt > 0 && col(UNITPRICE).toDouble >= 0 && col(CUSTOMERID).toString.trim() != "").map(col => {(col(CUSTOMERID).toString, col(QUANTITY).toInt \* BigDecimal(col(UNITPRICE)).setScale(2, BigDecimal.RoundingMode.HALF\_UP))})

val SumUQ = FilterMap.groupByKey().mapValues(sq => ((sq.sum/1000).toDouble.ceil)).map(col => {(col.\_2,col.\_1)})

val CLVCount = SumUQ.groupByKey().mapValues(sq => (sq.size))

val CLVInterval = CLVCount.map(col => {(col.\_1, (col.\_1\*1000).toString + "-" + ((col.\_1+1)\*1000).toString, col.\_2)}).sortBy(\_.\_1)

CLVInterval.map(r => r.productIterator.mkString("\t")).saveAsTextFile(OutputFile+"7")

##### Output:

