Fundamentals of Big Data Analytics
Project Phase 1 Report

Submitted to: Sir Kifayat Alizai

Submitted by:

Muhammad Owais

Muhammad Soban

Wasif Mehboob

Muhammad Shamil Umar

Table of Content

Main Objective:

For this phase, we were supposed to preprocess our data and store it so that in the next phase we can work on training models for our recommendation system.

Implementation:

Storing data:

There were alot of issues with storing data. One of the main issues was that some of the rows had different numbers of values like for example if the first line had 6 columns other lines might have 9 or 10 columns. So of course we had to define a structure. Secondly we had to decide which columns are important. We had such a large dataset which meant storage was a big issue.
So we used a software LTFViewer (Large Text File Viewer) to get a general idea how data was arranged. We noticed that following columns were most repeated:

- overall
- verified
- reviewerID
- asin
- reviewerName
- reviewText
- unixReviewTime
- vote

Which means only these columns were necessary to store. Secondly there were few columns which were repeated in some rows like scent was present twice in the same record which threw errors so defining a complete structure was very important so that all the unwanted rows could be neglected. So our scheme is looking like:

```
df.printSchema()
```

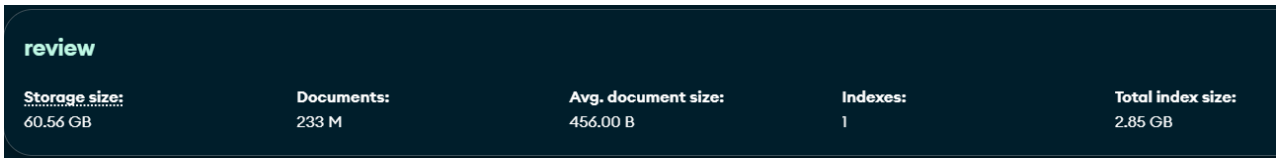
```
root
 |-- overall: float (nullable = true)
 |-- verified: boolean (nullable = true)
 |-- reviewerID: string (nullable = true)
 |-- asin: string (nullable = true)
 |-- reviewerName: string (nullable = true)
 |-- reviewText: string (nullable = true)
 |-- unixReviewTime: string (nullable = true)
 |-- vote: string (nullable = true)
```

And below is what our data frame looks like.

```
df = spark.read.json(r"C:/bda/All_Amazon_Review.json", schema = dataframe_format)
df.show(8)
```

```
+-------+--------+--------------+----------+---------------+--------------------+--------------+----+
|overall|verified|    reviewerID|      asin|   reviewerName|          reviewText|unixReviewTime|vote|
+-------+--------+--------------+----------+---------------+--------------------+--------------+----+
|    1.0|   false|A27BTSGLXK2C5K|B01709P72A|Jacob M. Wessler|Alexa is not able...|    1449792000|null|
|    4.0|   false|A27ZJ1NCBFP1HZ|B01709P72A|           Greg|Alexa works great...|    1449532800|   5|
|    1.0|   false| ACCQIOZMFN4UK|B01709P72A|       Da-Gr8-1|Weak!!\n\nAlexa d...|    1449446400|  11|
|    2.0|   false|A3KUPJ396OQF78|B01709P72A|  Larry Russlin|Can only control ...|    1449273600|null|
|    1.0|   false|A1U1RE1ZI19E1H|B01709P72A|        Rebekah|this worked great...|    1517529600|   2|
|    5.0|   false|A3TXR8GLKS19RE|B01709P72A|          Nello|         Great skill|    1515974400|null|
|    1.0|   false| AVIWE1LJXCG77|B01709P72A|   Pete Johnson|Pretty crappy. Wo...|    1515110400|   4|
|    1.0|   false|A1FOHYK23FJ6CN|B01709P72A|L. Ray Humphreys|Not happy. Can no...|    1515024000|   2|
+-------+--------+--------------+----------+---------------+--------------------+--------------+----+
only showing top 8 rows
```

Next thing we had to do was to decide on data types which we will store in mongodb. As you can see above, the vote is in string form so we converted it to integer. Another this to note is that even though unixReviewTime columns contains all numbers, we still decided to keep it as string because if we were to convert it into integer, it would convert to long integer data type which takes 8 bytes per value to store so we kept it as string so it can be stored in 1 to 4 bytes depending on size. After that we simply saved it in mongodb. This way we reduced our dataset size from 118GB to 60GB:



Each row is saved as separate record with all the columns in it:


```
_id: ObjectId('644edb388c1526367e424cee')
overall: 5
verified: false
reviewerID: "A13HK4LHAOXYRT"
asin: "B000068NW9"
reviewerName: "Slinky"
reviewText: "Great value in lower priced cable. Well constructed with quality conne…"
unixReviewTime: "1244332800"
vote: 8
```

EDA:

Next comes exploratory data analysis. We loaded data from mongodb for faster execution. We dropped the "_id" column and started working on handling NaN values. After exploring various algorithms we collectively decided to opt for collaborative filtering algorithms. It would require less columns to be processed and it would be faster. We will be using Locality Sensitive Hashing to find similar buyers and then we use cosine similarity to find items we need to recommend but before implementing we need to have insights about our data set. Using complete dataset is not possible for us. We have a lot of rows to process hence it is not possible to perform EDA on whole data unless we have multi node setup which unfortunately is not possible for us at the moment. One thing we can do is to do random sampling and in order to avoid random data to be biased we need to make sure that description of original dataset and sampled dataset resembles and luckily after 2 tries we were able to achieve this.

Original dataset:

```
result.show()
```

summary	asin	overall	reviewText	reviewerID	reviewerName	unixReviewTime	vote
count	233055327	233055327	232901943	233055327	233042194	233055327	32587701
mean	1.1606255546475441E9	4.232190208636595	Infinity	null	NaN	1.428970504369150 2E9	7.527312497435766
stddev	1.200537895407047E9	1.2465116112161108	NaN	null	NaN	8.577685647840 35E7	20.432629014560085
min	0000000116	0.0	\n	A0000040I1OM9N4SGBD8	\b T. T.	1000080000	2
max	B01HJI17Y8	5.0	When I pick up...	AZZZZXVAOWME		999993600	999

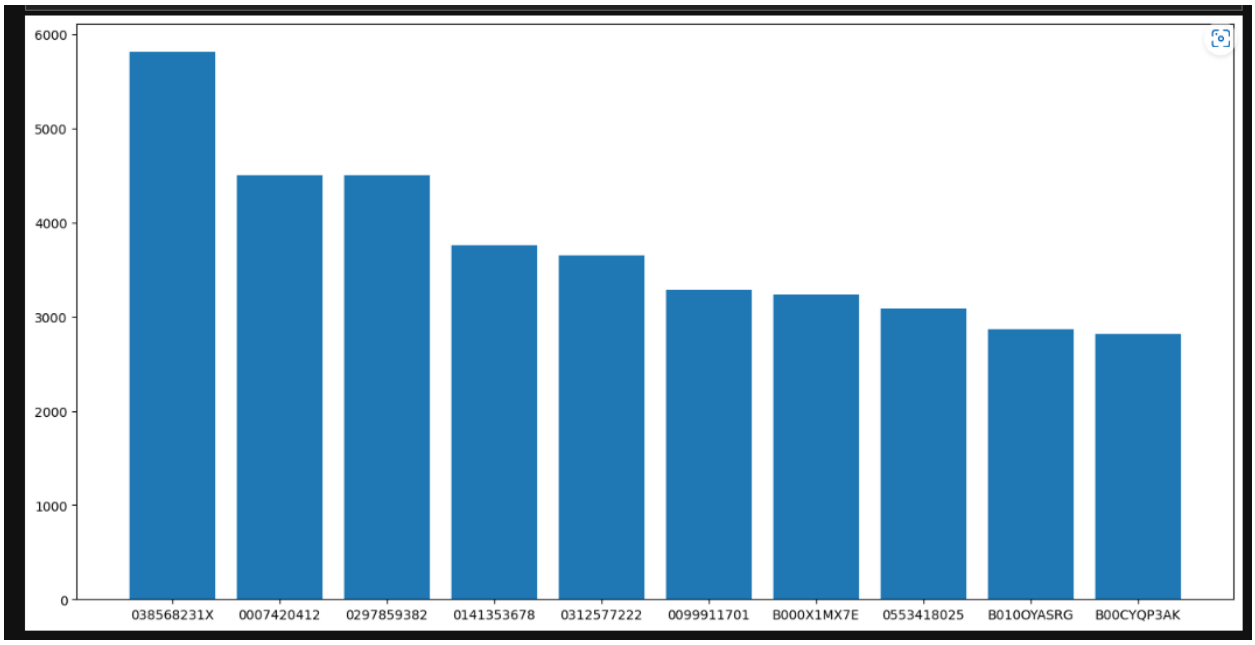
Sampled dataset:

```
df.select("asin", "overall","unixReviewTime","verified","vote").summary().show()
```

summary	asin	overall	unixReviewTime	vote
count	23311439	23311342	23310572	3258554
mean	1.1606514592942953E9	23465.592897611816	1.428971244624972E9	7.5228334408452335
stddev	1.1998400351548 63E9	5777483.874181405	8.575473616335726E7	20.4158658542515
min	's"	1.0	1000080000	2
25%	4.25284689E8	4.0	1.4028768E9	2
50%	9.80017106E8	5.0	1.446336E9	3
75%	1.509211748E9	5.0	1.4831424E9	6
max	zj"	1.5355872E9	999993600	997

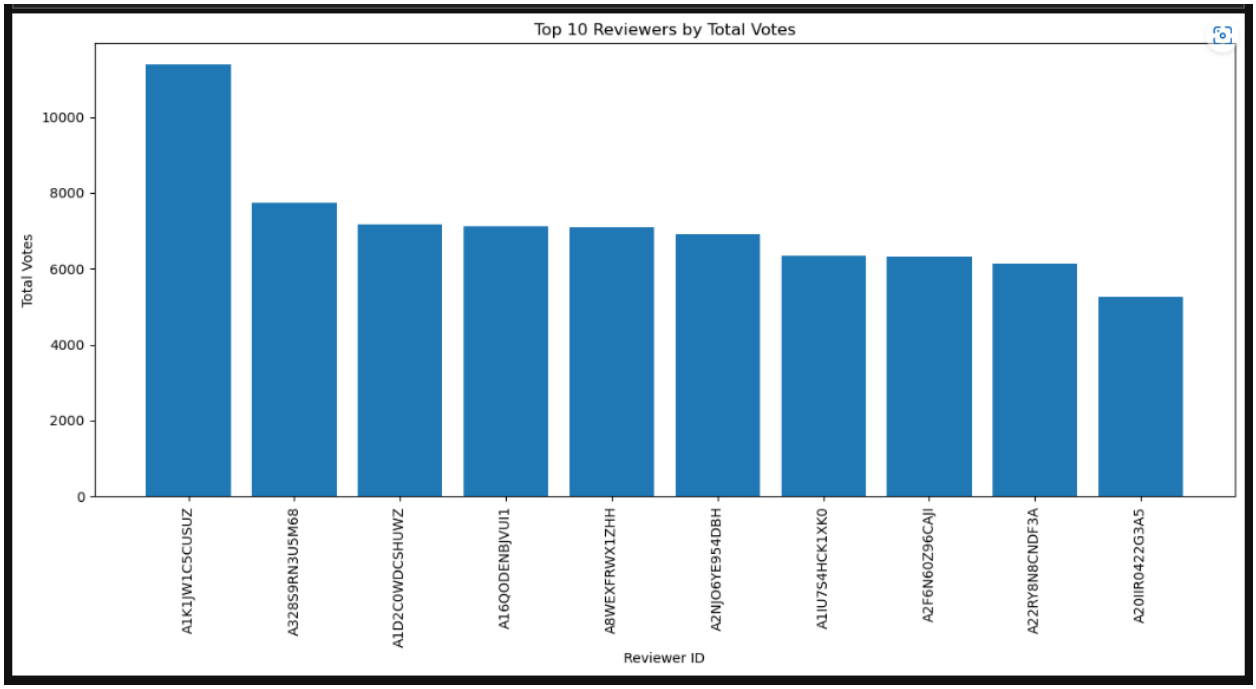
For processing we used the DASK module for python. It works like pyspark but it is designed to process data on a single machine unlike pyspark which highly depends on multi-clustering setup. Using this we were able to make some solid assumptions:

- Most Bought Items:



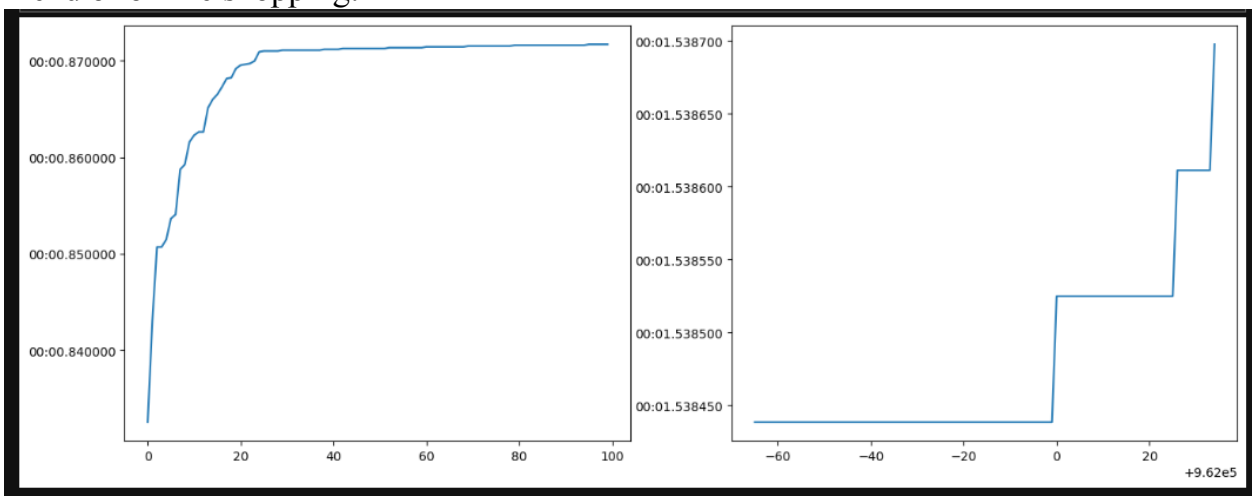
  Graph above tells us that these Products are most sold during this period and it tells us that they are most likely to be recommended to users. Even if cosine similarity between 2 users doesn't fall inside the threshold, we can safely recommend these products.

- Trusted Reviewers:



Above graph tells us that top 10 reviewers with these IDs are most trusted because they have the most upvotes during the period in which data was collected so we can assume that reviews they provided are accurate and people tend to agree with them. We will have to consider this while training our model.

- Trend of online shopping:



We graph on left tell us number products that are reviewed at start of data collection and right graph tells us number products that are reviewed at ending of data collection. This tells us that people are doing online shopping more than ever and that is probably because of increasing accessibility of the internet around the world. These graphs might not be accurate as not all people post their reviews but it does give us a strong idea that our proposition is true. This means that products that are bought later are most likely reviewed accurately as more people are dropping reviews on each product.

Whats Next:

We will extract reviewerID, asin and overall columns and we will design a matrix. Reviewers as rows, asin (Product IDs) as columns and store overall values in corresponding cells. Then we will be designing an LSH algorithm and will integrate it in our flask application.

Contributions:

We divided our groups in 2. Theory crafting and coding. Muhammad Owais, Muhammad Soban and Wasif Mehboob were in theory a crafting group. They worked on understanding algorithms like collaborative filtering, content-based filtering, matrix factorization and also to explore other algorithms. On the other hand Muhammad Shamil Umar worked on understanding spark and how everything works in it, what other options do we have to process or data and coding.