# Amazon Electronics Products Analysis

By,

Sushmanth Sagala

Hadoop Project – UpX Academy

# Project Information

- **Domain:** Retail
- Technology use: Hadoop, Hive
- Dataset: <a href="http://jmcauley.ucsd.edu/data/amazon/links.html">http://jmcauley.ucsd.edu/data/amazon/links.html</a>
- The Amazon Product data set to be considered for this project is from Amazon, a well-known e-commerce giant. This is a very interesting project to get insights based analysis of product sales based on reviews and ratings.
- This dataset contains product reviews including 142.8 million reviews spanning May 1996 July 2014. This data set includes reviews (ratings, text, helpfulness votes). From the various categories the Electronics category is chosen for the analysis. The size of reviews data set is 1.5 GB and ratings data set is 320 MB.

# Data Description

#### Review:

- reviewerID ID of the reviewer, e.g. A2SUAM1J3GNN3B
- asin ID of the product, e.g. 0000013714
- reviewerName name of the reviewer
- helpful helpfulness rating of the review, e.g. 2/3
- reviewText text of the review
- overall rating of the product
- summary summary of the review
- unixReviewTime time of the review (unix time)
- reviewTime time of the review (raw)

#### Rating:

- user ID of the reviewer, e.g. A2SUAM1J3GNN3B
- Item ID of the product, e.g. 0000013714
- Rating rating of the product
- Timestamp time of the review (unix time)

### **Business Questions**

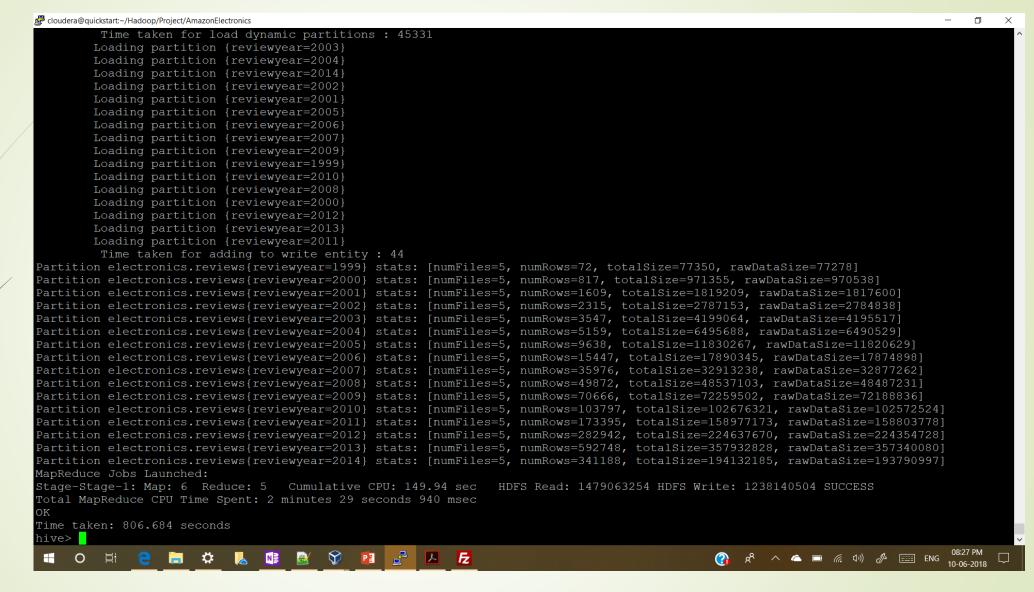
#### Hadoop / Hive

- Understand the Dataset
- Top electronic item that was rated above 3.0 over the period of 8 years [2006-2014] (EG. Year 2006 100 items bought received all the rating above 3.0 show top 1 item out of 100; sample data 2006, "ASIN", 'Review Count")
- Worst review a product received in a given year between 2006 to 2014 (Eg. 2006, ASIN, Review Count (Least number of reviews and below 2.0 rating)
- Maximum number of reviews given by same user in a year
- Least Helpful reviews per year per products (helpful percentage between 1% and 30 %)
- Most Helpful reviews Per year Per Products (Helpful percentage >75%)
- Growth of review comments on products/year
- Visualization of results observe

### Q1. Understand the Dataset

#### Reviews dataset:

- Reviews dataset brought to HDFS as a Hive table using org.apache.hive.hcatalog.data.JsonSerDe library to load Json data.
  - CREATE TABLE amazon\_reviews (reviewerID string, asin string, reviewerName string, helpful array<int>, reviewText string, overall float, summary string, unixReviewTime int, reviewTime string)ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerDe' STORED AS TEXTFILE;
  - LOAD DATA LOCAL INPATH 'data/Reviews/Electronics\_5.json' OVERWRITE INTO TABLE amazon\_reviews;
- Partitioned reviews data based on review year and clustered the data based on review rating.
  - CREATE TABLE reviews (reviewerID string, asin string, reviewerName string, helpful array<int>, reviewText string, overall float, summary string, unixReviewTime int, reviewTime string) PARTITIONED BY (reviewYear int) CLUSTERED BY (overall) INTO 5 BUCKETS ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
  - INSERT INTO TABLE reviews PARTITION(reviewYear) SELECT reviewerID, asin, reviewerName, helpful, reviewText, overall, summary, unixReviewTime, reviewTime, year(from\_unixtime(unix\_timestamp(reviewTime, 'MM dd, yyyy'))) FROM amazon\_reviews;

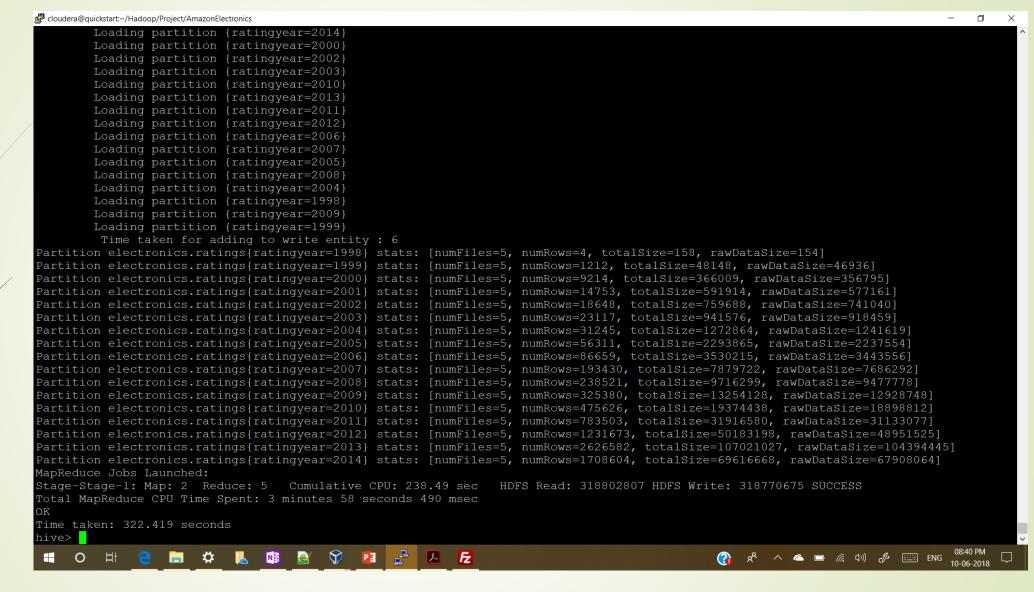


Sample output screenshot

### Q1. Understand the Dataset

#### Ratings dataset:

- Ratings dataset brought to HDFS as a Hive table.
  - CREATE TABLE amazon\_ratings (user string, item string, rating float, timestamp int)
     ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' STORED AS TEXTFILE;
  - LOAD DATA LOCAL INPATH 'data/Ratings/ratings\_Electronics.csv' OVERWRITE INTO TABLE amazon\_ratings;
- Partitioned rating data based on rating year and clustered the data based on rating.
  - CREATE TABLE ratings (user string, item string, rating float, timestamp int) PARTITIONED BY (ratingYear int) CLUSTERED BY (rating) INTO 5 BUCKETS ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
  - INSERT INTO TABLE ratings PARTITION(ratingYear) SELECT user, item, rating, timestamp, year (date\_add(from\_unixtime(timestamp), 1)) FROM amazon\_ratings;



Sample output screenshot

- Q2. Top electronic item that was rated above 3.0 over the period of 8 years [2006-2014] (EG. Year 2006 100 items bought received all the rating above 3.0 show top 1 item out of 100; sample data 2006, "ASIN", 'Review Count")
  - To determine the Top electronics item rated above 3.0 against each year [2006-2014].
  - Used outer join as all reviewers will be present in ratings dataset. Right outer join is used as ratings is bigger table.
  - Grouped all the Items against each year with avg rating greater than 3.0 and choosing the item with highest review count against each year [2006-2014].

SELECT r.ratingYear,r.item,r.review count

FROM (SELECT rt.ratingYear,rt.item,rt.review\_count,ROW\_NUMBER() OVER(PARTITION BY rt.ratingYear ORDER BY rt.review\_count DESC) AS item\_rank

FROM (SELECT rat.ratingYear,rat.item,AVG(rat.rating) AS avg\_rating,COUNT(rev.reviewerlD) AS review\_count

FROM (SELECT reviewYear,asin,reviewerID FROM reviews WHERE reviewYear >= 2006 AND reviewYear <= 2014) rev

LEFT OUTER JOIN (SELECT ratingYear, item, user, rating FROM ratings WHERE ratingYear >= 2006 AND ratingYear <= 2014) rat

ON rev.asin = rat.item AND rev.reviewerID = rat.user

GROUP BY rat.ratingYear,rat.item) rt

WHERE avg rating > CAST(3.0 AS FLOAT) ) r

WHERE r.item rank = 1:

- Q3. Worst review a product received in a given year between 2006 to 2014 (Eg. 2006, ASIN, Review Count (Least number of reviews and below 2.0 rating)
  - To determine the worst electronics item rated below 2.0 against each year [2006-2014].
  - Used outer join as all reviewers will be present in ratings dataset. Right outer join is used as ratings is bigger table.
  - Grouped all the Items against each year with avg rating lesser than 2.0 and choosing the item with least review count against each year [2006-2014].

SELECT r.ratingYear,r.item,r.review\_count

FROM (SELECT rt.ratingYear,rt.item,rt.review\_count,ROW\_NUMBER() OVER(PARTITION BY rt.ratingYear ORDER BY rt.review\_count ASC) AS item\_rank

FROM (SELECT rat.ratingYear,rat.item,AVG(rat.rating) AS avg\_rating,COUNT(rev.reviewerID) AS review\_count

FROM (SELECT reviewYear,asin,reviewerID FROM reviews WHERE reviewYear >= 2006 AND reviewYear <= 2014) rev

RIGHT OUTER JOIN (SELECT ratingYear, item, user, rating FROM ratings WHERE ratingYear >= 2006 AND ratingYear <= 2014) rat

ON rev.asin = rat.item AND rev.reviewerID = rat.user

GROUP BY rat.ratingYear,rat.item) rt

WHERE avg\_rating < CAST(2.0 AS FLOAT) ) r

WHERE r.item\_rank = 1;

```
cloudera@quickstart:~/Hadoop/Project/AmazonElectronics
2018-06-10 08:56:07,610 Stage-2 map = 100%, reduce = 83%, Cumulative CPU 52.51 sec
2018-06-10 08:56:10,755 Stage-2 map = 100%, reduce = 95%, Cumulative CPU 55.59 sec
2018-06-10 08:56:12,915 Stage-2 map = 100%, reduce = 100%, Cumulative CPU 57.07 sec
MapReduce Total cumulative CPU time: 57 seconds 70 msec
Ended Job = job 1528528998925 0035
Launching Job 3 out of 3
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
 set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
 set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
 set mapreduce.job.reduces=<number>
Starting Job = job 1528528998925 0036, Tracking URL = http://quickstart.cloudera:8088/proxy/application 1528528998925 0036/
Kill Command = /usr/lib/hadoop/bin/hadoop job -kill job 1528528998925 0036
Hadoop job information for Stage-3: number of mappers: 1; number of reducers: 1
2018-06-10\ 08:56:34,369\ Stage-3\ map = 0\%, reduce = 0%
2018-06-10 08:56:51,369 Stage-3 map = 100%, reduce = 0%, Cumulative CPU 11.04 sec
2018-06-10\ 08:57:06,408\ Stage-3\ map = 100\%, reduce = 70%, Cumulative CPU 18.39 sec
2018-06-10 08:57:09,565 Stage-3 map = 100\%, reduce = 93\%, Cumulative CPU 22.12 sec
2018-06-10 08:57:11,697 Stage-3 map = 100%, reduce = 100%, Cumulative CPU 24.3 sec
MapReduce Total cumulative CPU time: 24 seconds 300 msec
Ended Job = job 1528528998925 0036
MapReduce Jobs Launched:
Stage-Stage-1: Map: 7 Reduce: 6 Cumulative CPU: 357.09 sec HDFS Read: 1522569315 HDFS Write: 101301379 SUCCESS
Stage-Stage-2: Map: 1 Reduce: 1 Cumulative CPU: 57.07 sec HDFS Read: 101309202 HDFS Write: 23098966 SUCCESS
Stage-Stage-3: Map: 1 Reduce: 1 Cumulative CPU: 24.3 sec HDFS Read: 23106125 HDFS Write: 182 SUCCESS
Total MapReduce CPU Time Spent: 7 minutes 18 seconds 460 msec
       B000CS7U1C
       B000BKJZ9Q
                       192
                       270
       B000LRMS66
                       272
       B0002L5R78
       B001T9NUJE
                       260
                       527
       B0019EHU8G
       B0019EHU8G
2013
       B007WTAJTO
                       2678
       B00DR0PDNE
Time taken: 750.15 seconds, Fetched: 9 row(s)
hive>
                                                                                             (3) x<sup>8</sup> ∧ △ □ (6, □)) (4) === ENG 10-06-2018 □
💶 O 🛱 🤚 🐚 🗘 👢 🐠 遂 😯 📴 💤 🔼 左
```

Sample output screenshot

- Q4. Maximum number of reviews given by same user in a year
  - To determine the reviewer by whom maximum number of reviews given in a year.
  - Used outer join as all reviewers will be present in ratings dataset. Right outer join is used as ratings is bigger table.
  - Grouped all the reviewers against each year to find the highest reviews given.

SELECT rev.reviewYear,rev.reviewerID,COUNT(rev.reviewerID) AS review\_count

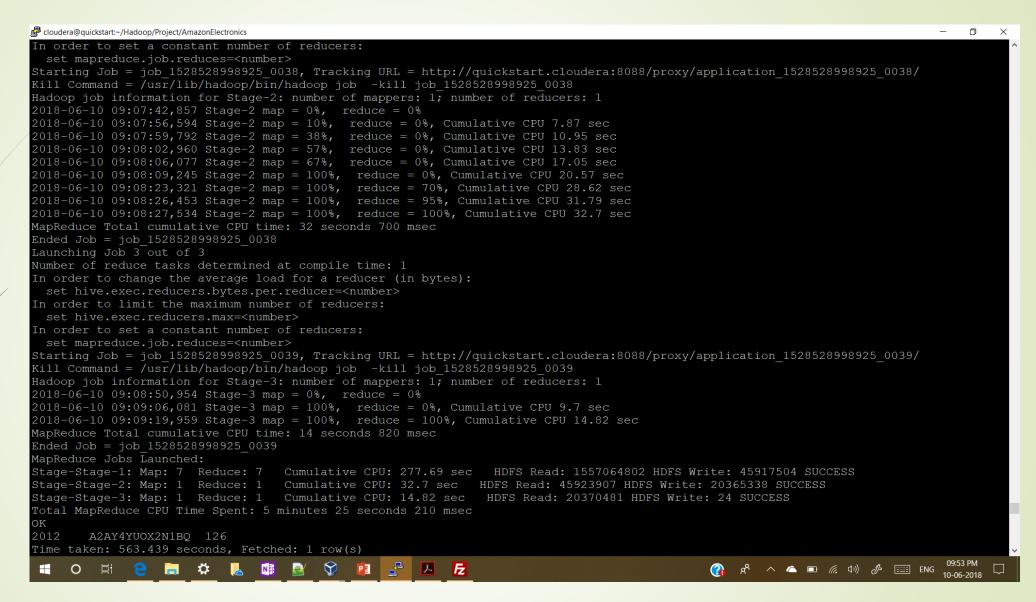
FROM reviews rev

RIGHT OUTER JOIN ratings rat

ON rev.asin = rat.item AND rev.reviewerID = rat.user

GROUP BY rev.reviewYear,rev.reviewerID

ORDER BY review\_count DESC LIMIT 1;



Sample output screenshot

- Q5. Least Helpful reviews per year per products (helpful percentage between 1% and 30 %)
  - To determine the Item per year with least helpful reviews within 1% and 30%.
  - Used outer join as all reviewers will be present in ratings dataset. Right outer join is used as ratings is bigger table.
  - Grouped all the Items against each year with avg helpful percentage between 1% and 30%, choosing the item with least helpful reviews against each year.

SELECT r.ratingYear,r.item,r.avg\_helpful\_perc

FROM (SELECT rt.ratingYear,rt.item,rt.avg\_helpful\_perc,ROW\_NUMBER() OVER(PARTITION BY rt.ratingYear ORDER BY rt.avg\_helpful\_perc ASC) AS helpful\_perc\_rank

FROM (SELECT rat.ratingYear,rat.item,AVG((helpful[0] / helpful[1]) \* 100) AS avg\_helpful\_perc

FROM reviews rev

RIGHT OUTER JOIN ratings rat

ON rev.asin = rat.item AND rev.reviewerID = rat.user

GROUP BY rat.ratingYear,rat.item) rt

WHERE rt.avg\_helpful\_perc >= CAST(1.0 AS FLOAT) AND avg\_helpful\_perc <= CAST(30.0 AS FLOAT) r

WHERE r.helpful\_perc\_rank = 1;

- Q6. Most Helpful reviews Per year Per Products (Helpful percentage >75%)
  - To determine the Item per year with most helpful reviews above 75%.
  - Used outer join as all reviewers will be present in ratings dataset. Right outer join is used as ratings is bigger table.
  - Grouped all the Items against each year with avg helpful percentage above 75%, choosing the item with most helpful reviews against each year.

```
SELECT r.ratingYear,r.item,r.avg_helpful_perc
```

FROM (SELECT rt.ratingYear,rt.item,rt.avg\_helpful\_perc,ROW\_NUMBER() OVER(PARTITION BY rt.ratingYear ORDER BY rt.avg\_helpful\_perc DESC) AS helpful\_perc\_rank

FROM (SELECT rat.ratingYear,rat.item,AVG((helpful[0] / helpful[1]) \* 100) AS avg\_helpful\_perc

FROM reviews rev

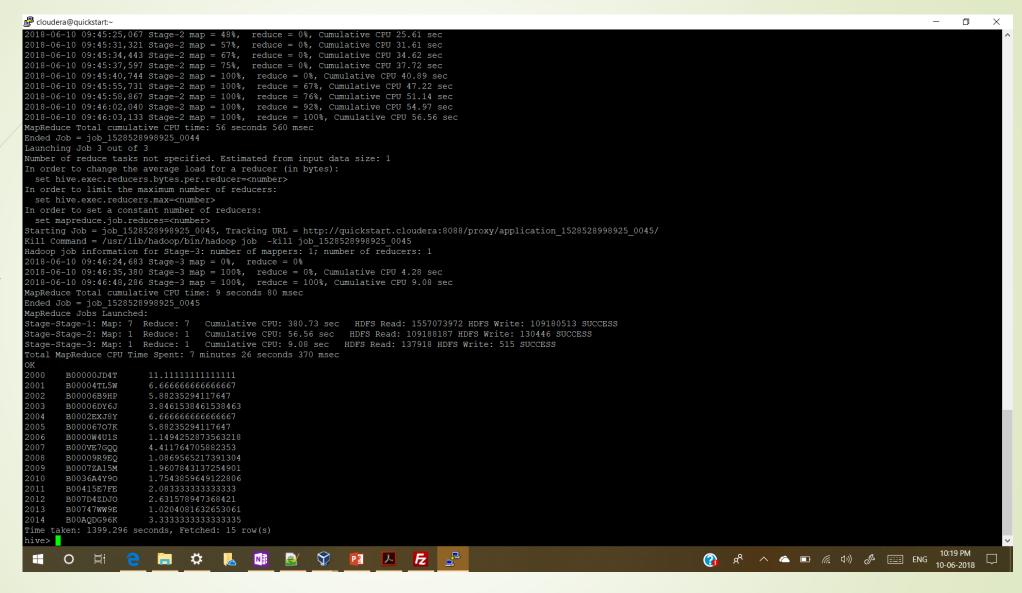
RIGHT OUTER JOIN ratings rat

ON rev.asin = rat.item AND rev.reviewerID = rat.user

GROUP BY rat.ratingYear,rat.item) rt

WHERE rt.avg\_helpful\_perc > CAST(75.0 AS FLOAT) ) r

WHERE r.helpful\_perc\_rank = 1;



Sample output screenshot

- Q7. Growth of review comments on products/year
  - To determine the growth of the review comments against each product every year.
  - Transposed review count of each year against every product using Map function and group by.
  - Grouped each product with a map of review counts against each year.

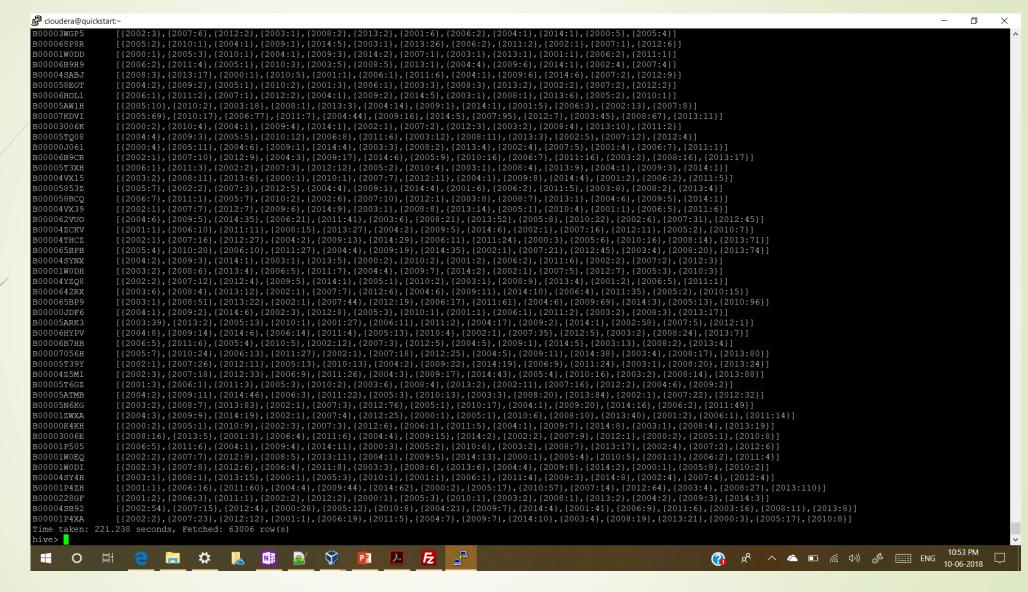
SELECT r.asin,collect\_list(r.year\_count\_map) AS Year\_count

FROM (SELECT rev.asin,map(rev.reviewYear,COUNT(rev.reviewerID)) AS year\_count\_map

FROM reviews rev

GROUP BY rev.reviewYear,rev.asin) r

GROUP BY r.asin:



Sample output screenshot

- Q8. Visualization of results observe
  - Reviews data is a subset of ratings dataset with additional details of review comments, helpful, summary, etc,.
  - Reviews adoption improved drastically from year 2007-2008.
  - Ratings adoption improved drastically from year 2006-2007.
  - Number of reviewer's comments per year started to multiple from year 2007, but in 2014 there was a drop in number unique reviewers.
  - Number of users per year started to multiple from year 2004, but in 2014 there was drop in number unique users.
  - As a trend number items reviewed each year is directly proportional to the number of users, same trend can be seen even for the drop in number of reviews during year 2014.

### Conclusion

- Amazon Electronics Products data loaded and analysed successfully.
- Created an Database in Hive and loaded Reviews/Rating data as hive tables.
- Partitioned the Hive tables to answer the business questions using Hive queries.
- Code: <a href="https://github.com/ssushmanth/AmazonElectronicsAnalysis-Hive">https://github.com/ssushmanth/AmazonElectronicsAnalysis-Hive</a>