**HIVE CASE STUDY**

**PROBLEM STATEMENT:**

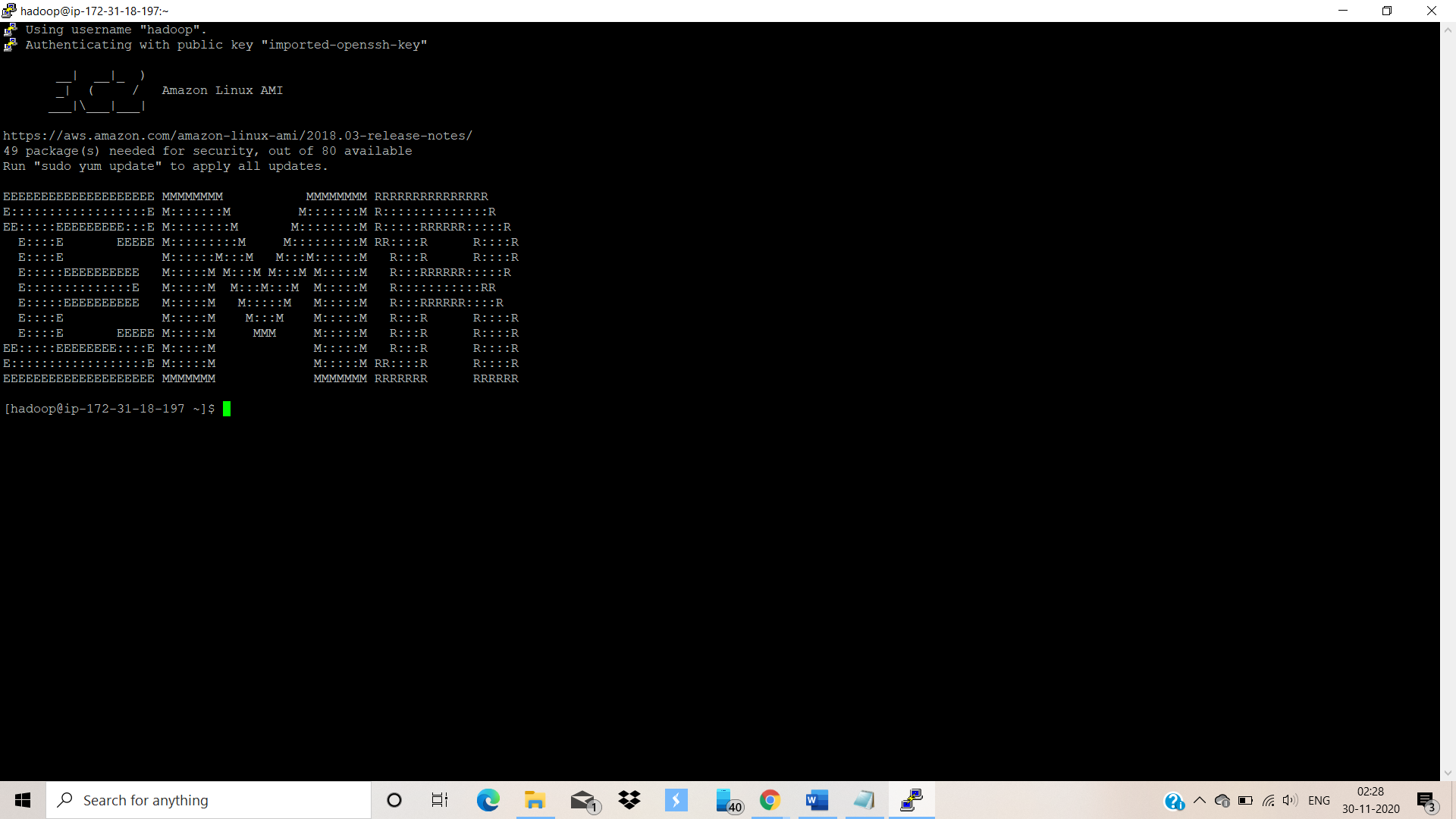
With online sales gaining popularity, tech companies are exploring ways to improve their sales by analysing customer behaviour and gaining insights about product trends. Our task is to extract data and gather insights from a real-life data set of an e-commerce company.

**DATASET GIVEN:**

For this assignment, we will be working with a public clickstream dataset of a cosmetics store. Using this dataset, we will try to extract valuable insights.

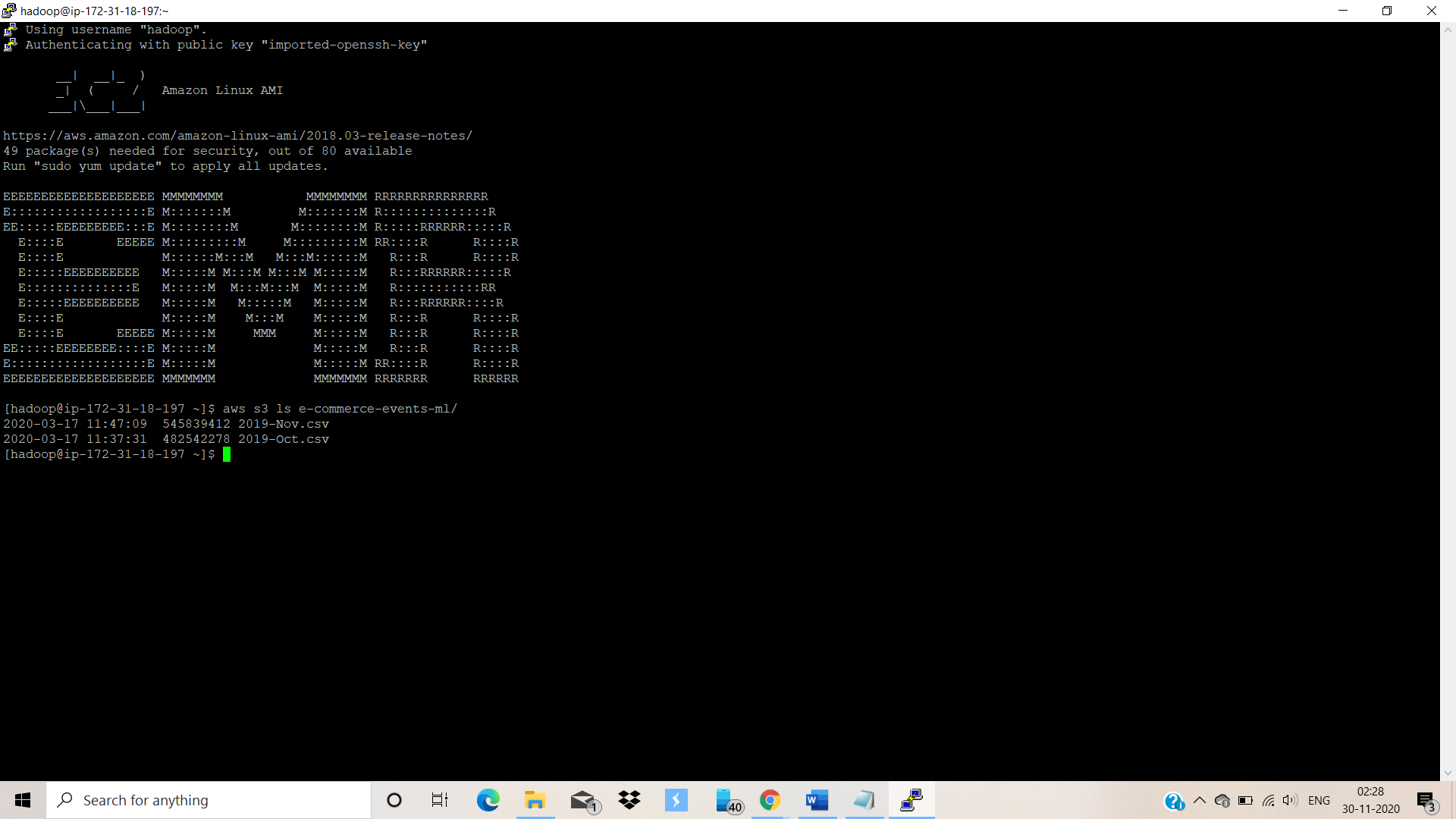
**IMPLEMENTATION:**

* **Firstly we launched an EMR cluster that utilizes the Hive services as shown :**



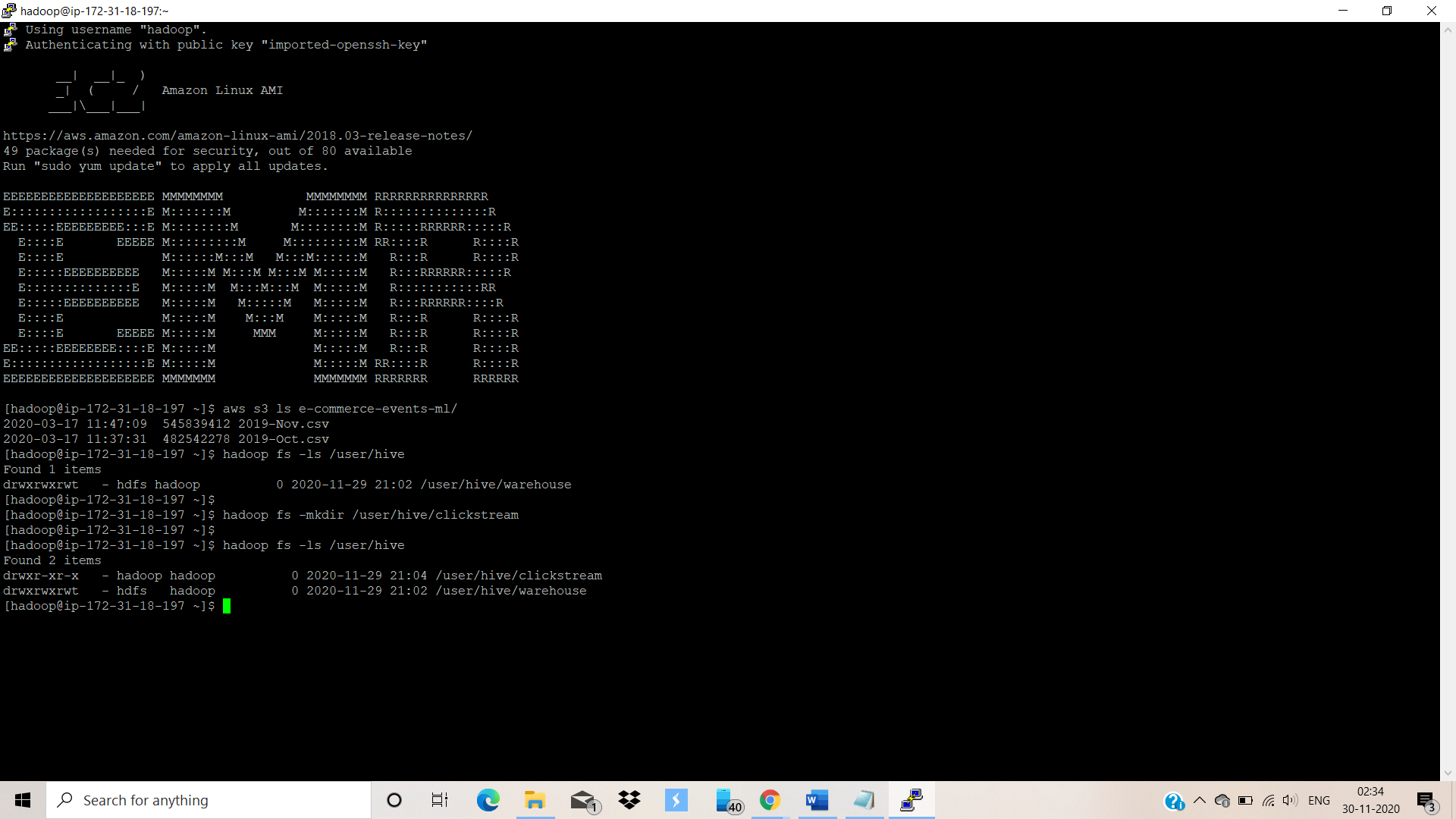
* **Checking the data files which are loaded into S3 by using the following command :**

aws s3 ls e-commerce-events-ml/



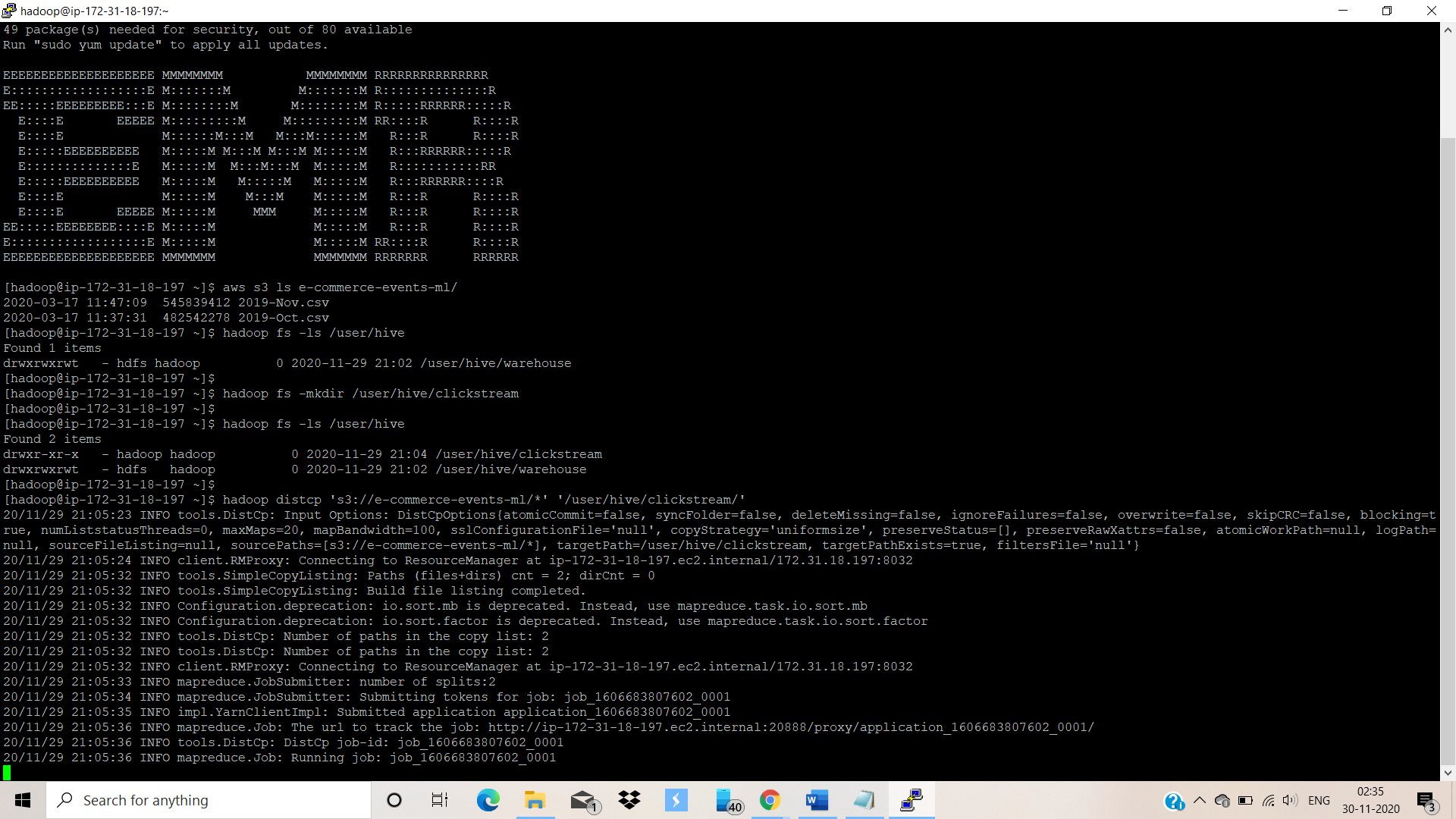
* **create a directory named “clickstream” in HDFS to collect the input data:**

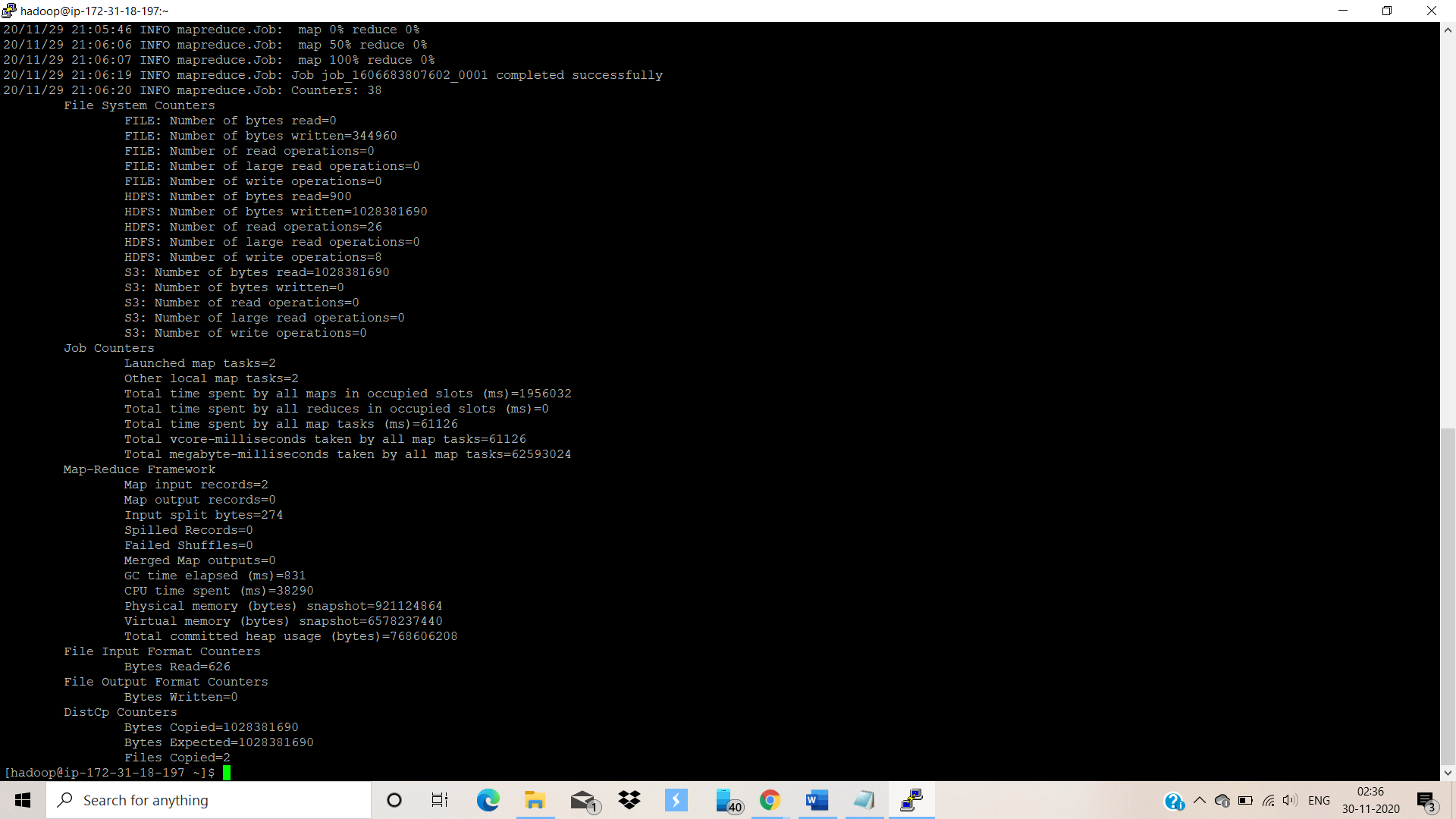
hadoop fs -mkdir /user/hive/clickstream



* **Move the data from the S3 bucket into the HDFS :**

hadoop distcp 's3://e-commerce-events-ml/\*' '/user/hive/clickstream/'





* **Launching HIVE service and Creating the structure of our table :**

We are going to create an **EXTERNAL table** .

**Reason for creating an external table :** even if delete the external table , only the metadata gets deleted ,the data doesn’t get deleted , it remains intact in the HDFS so that we do not need to load the data again while using it again.

**Creating a table named**: **clickstream\_info** with the following structure :

CREATE EXTERNAL TABLE IF NOT EXISTS clickstream\_info(

event\_time string,

event\_type string,

product\_id string,

category\_id string,

category\_code string,

brand string,

price float,

user\_id bigint,

user\_session string)

ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' lines terminated by '\n'

LOCATION '/user/hive/clickstream/'

tblproperties("skip.header.line.count"="1") ;

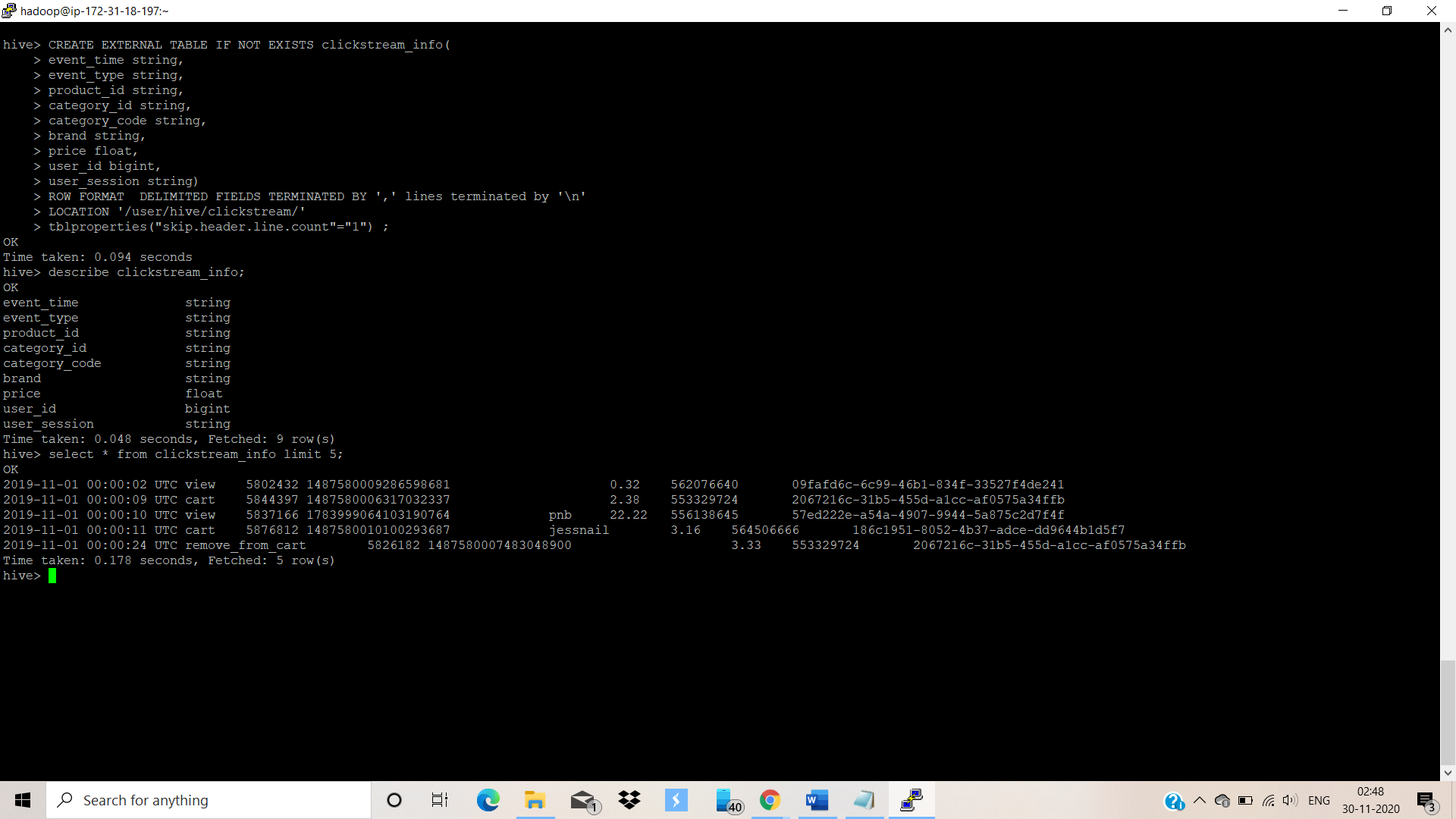
* **Looking at the table description:**

describe clickstream\_info;

* **Looking at the top 5 rows to check whether the data is loaded properly.**

select \* from clickstream\_info limit 5;

**Above queries have been shown below:**



**So from the above screenshot, we can see that the data is loaded properly .**

**CREATION OF AN OPTIMIZED TABLE:**

**We are going to use optimization techniques like partitioning and bucketing to increase the performance of the queries ie. to reduce the time taken for the queries to execute .**

* **To enable partitioning and bucketing:**

set hive.exec.dynamic.partition.mode=nonstrict;

set hive.exec.dynamic.partition=true;

set hive.enforce.bucketing=true ;

* **Creating a Partitioned and Bucketed Table named** : **clickstream\_part :**

CREATE TABLE IF NOT EXISTS clickstream\_part(

event\_time string,

product\_id string,

category\_id string,

category\_code string,

brand string,

price float,

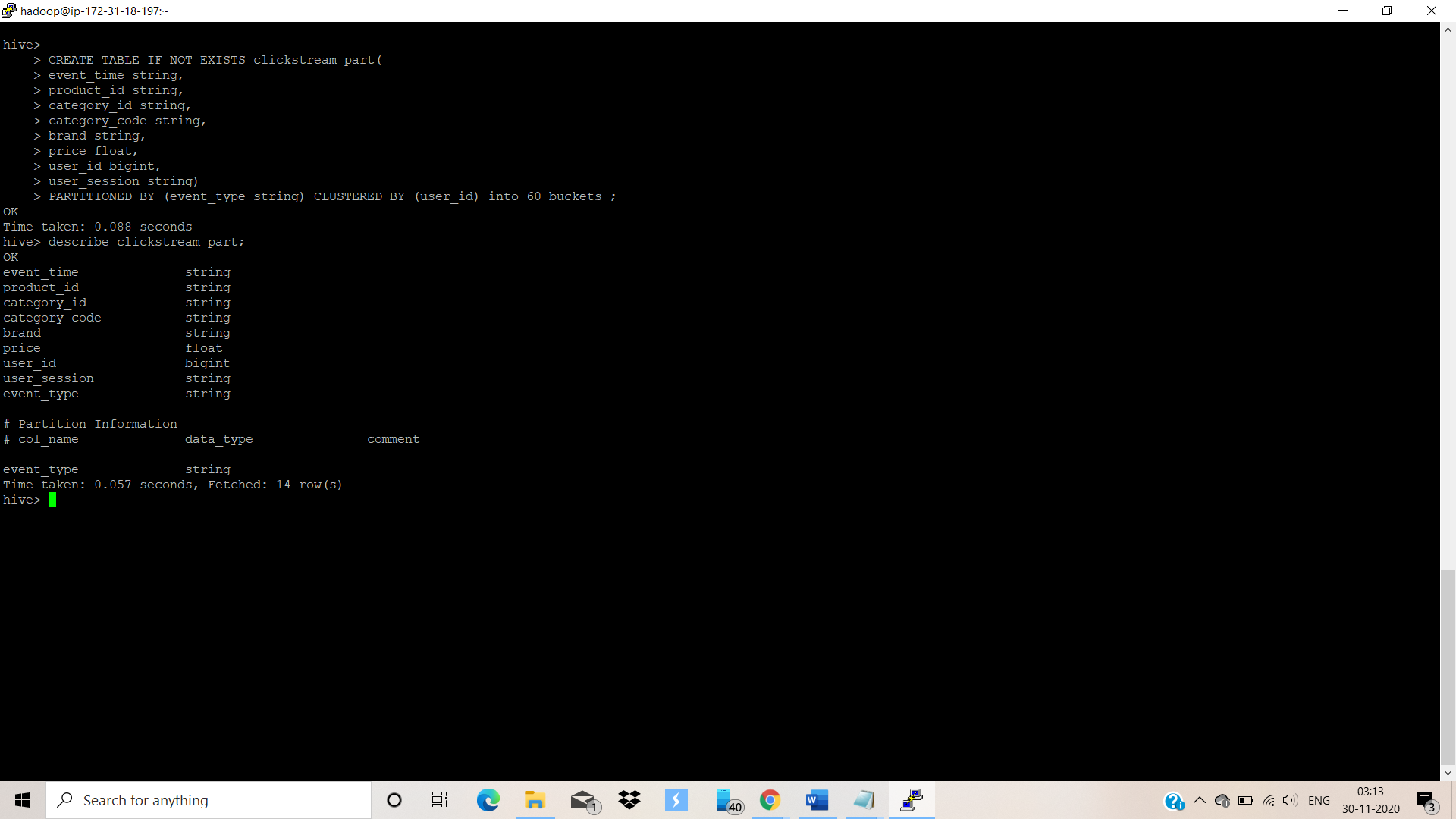
user\_id bigint,

user\_session string)

PARTITIONED BY (event\_type string) CLUSTERED BY (user\_id) into 60 buckets ;

* **Looking at the optimized table description**

describe clickstream\_part;



* **To load the optimized hive table:**

insert into table clickstream\_part partition(event\_type) select

event\_time ,

product\_id ,

category\_id ,

category\_code,

brand ,

price ,

user\_id ,

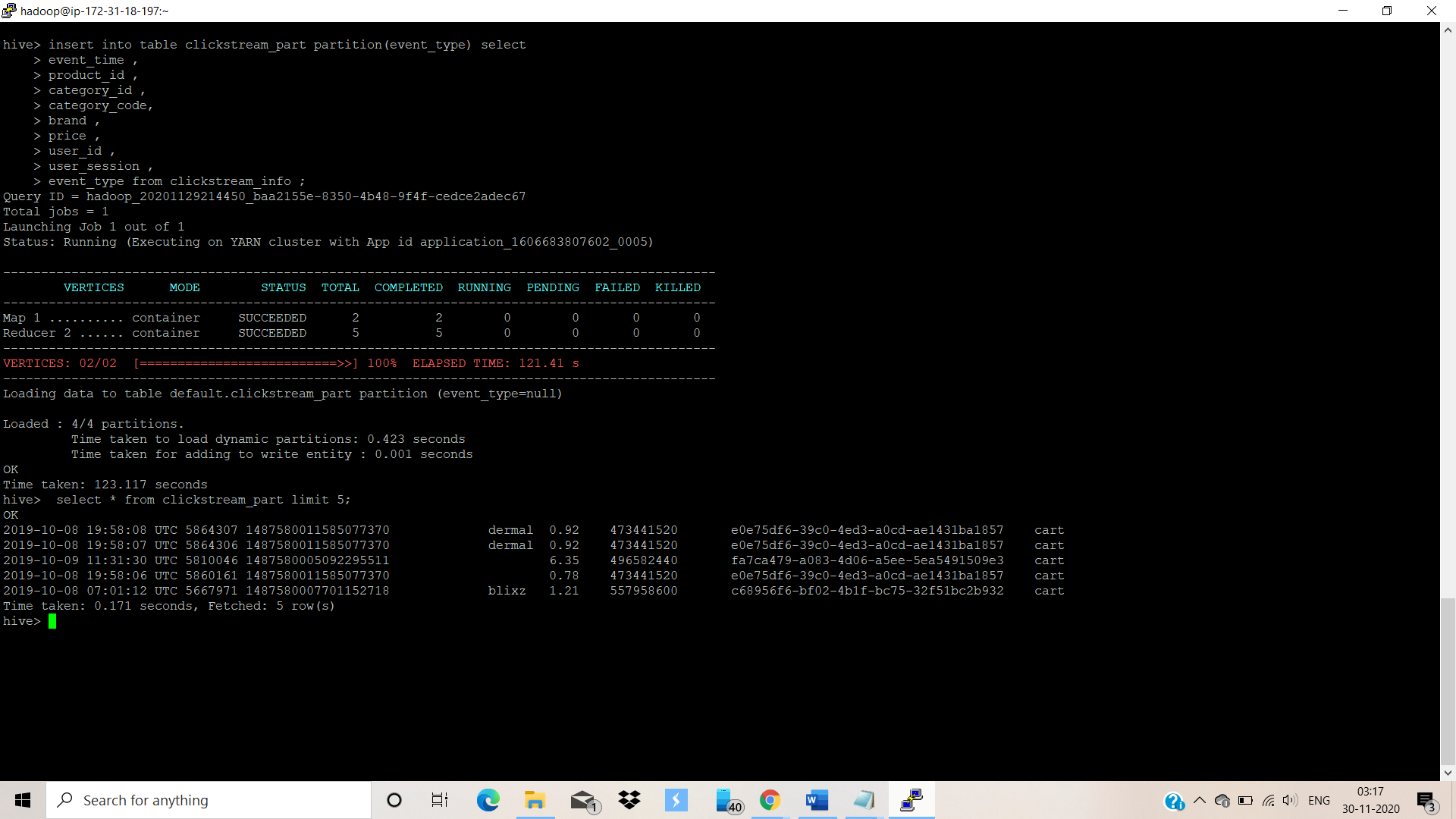
user\_session ,

event\_type from clickstream\_info ;

* **Looking at the top 5 rows to check whether the data is loaded properly.**

select \* from clickstream\_part limit 5;

**Above queries have been shown below:**



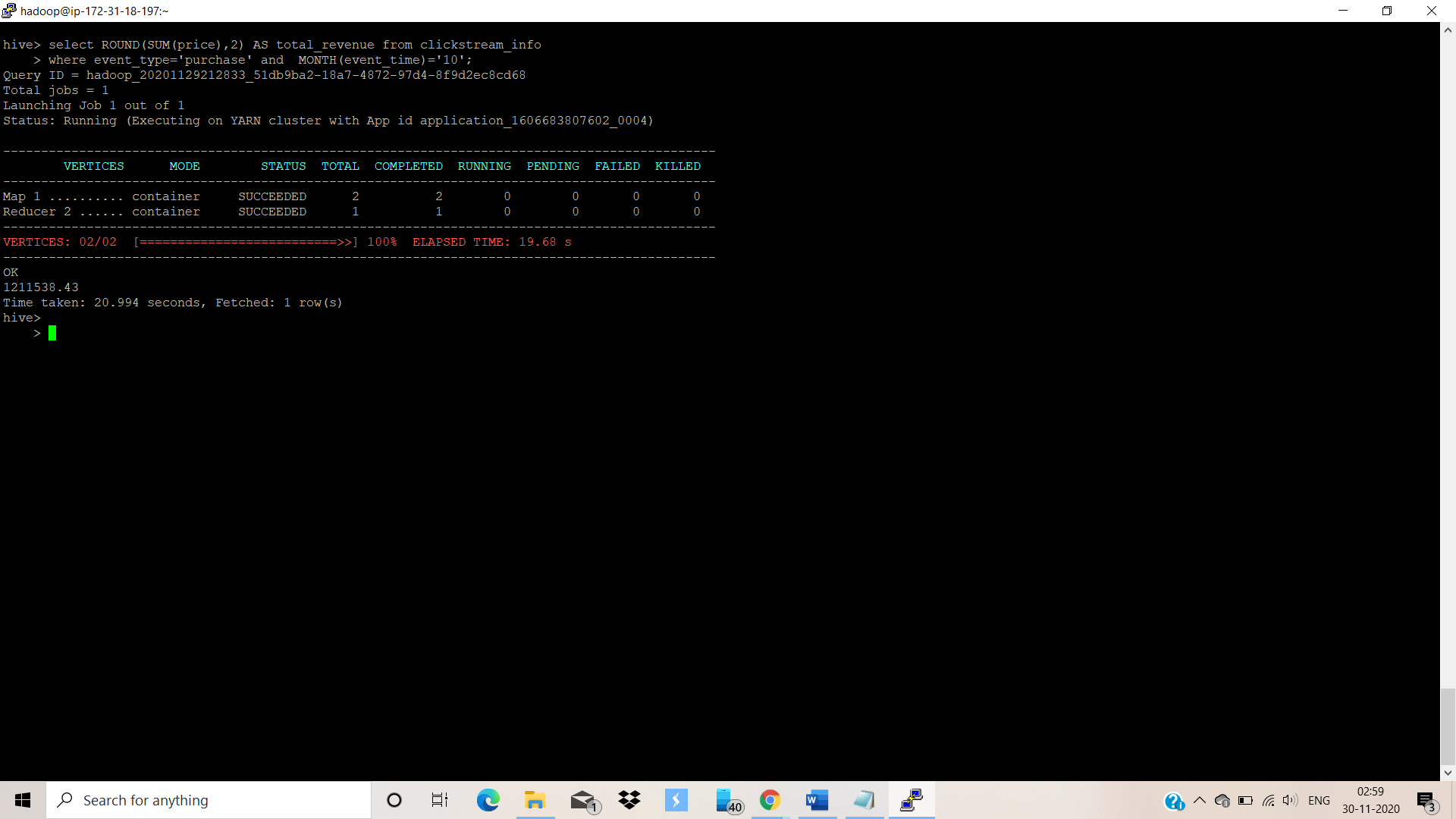
**So from the above screenshot, we can see that the data is loaded properly into the optimized table .**

**PERFORMANCE COMPARISON BETWEEN THE NORMAL TABLE AND THE OPTIMIZED TABLE:**

Now we will run the same query on both normal and optimized tables and see which one performs better:

* **Running the query on normal table : clickstream\_info**
  + - Find the total revenue generated due to purchases made in October.
* Query🡪 select ROUND(SUM(price),2) AS total\_revenue from **clickstream\_info**

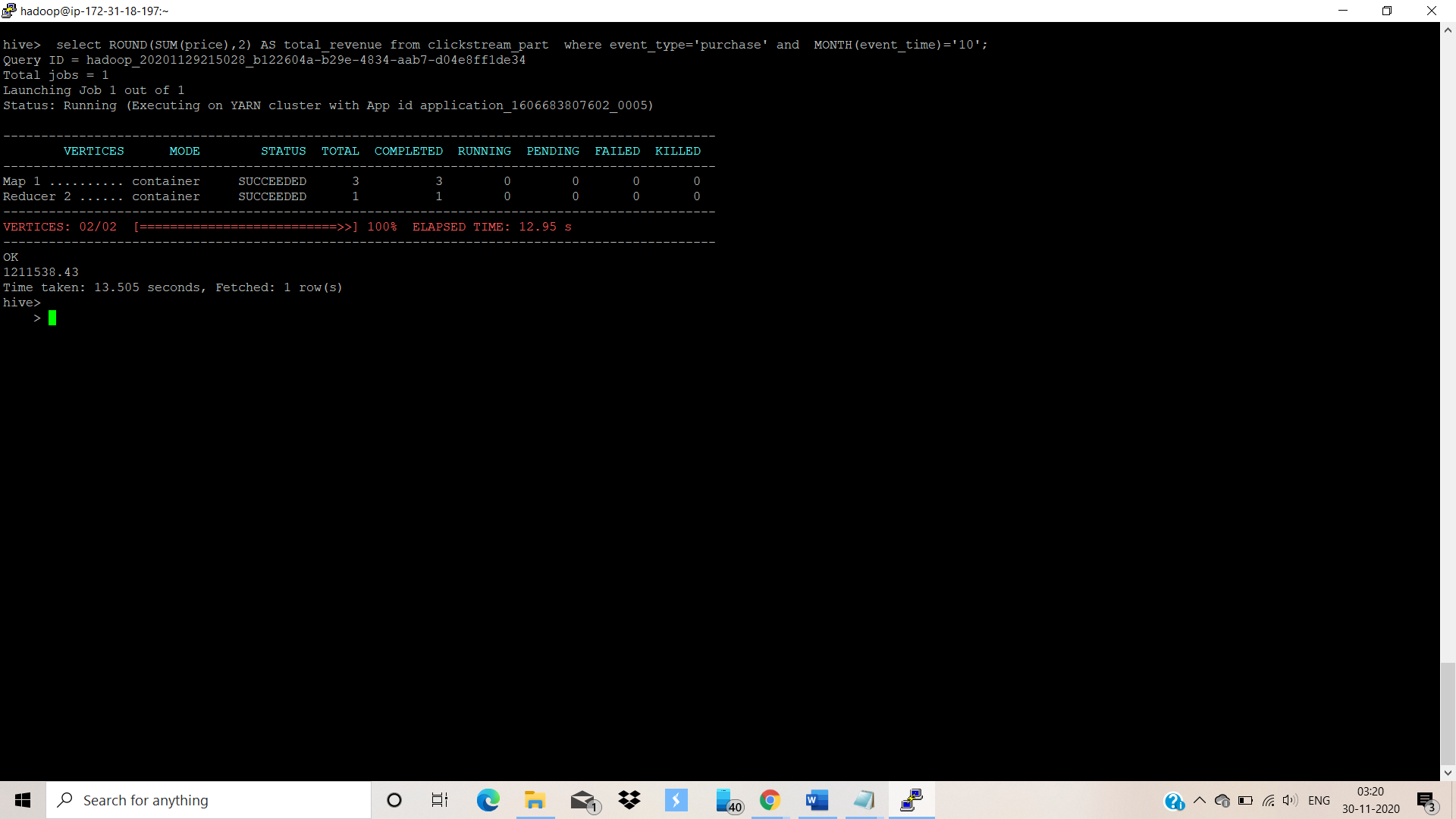
where event\_type='purchase' and MONTH(event\_time)='10';



Time taken to execute the above query by **normal table is : 20.994 seconds**.

* **Running the query on optimized table : clickstream\_part**
  + - Find the total revenue generated due to purchases made in October.

Query🡪 select ROUND(SUM(price),2) AS total\_revenue from **clickstream\_part** where event\_type='purchase' and MONTH(event\_time)='10';



Time taken to execute the above query by **optimized table is : 13.505 seconds.**

**INFERENCE: On comparing both the query execution times , we can conclude that the Optimized table takes lesser time to execute the same query as compared to the time taken by the normal table . Therefore , we will be using the Optimized Table named : clickstream\_part in our further analysis .**

**RUNNING HIVE QUERIES TO ANSWER THE FOLLOWING QUESTIONS:**

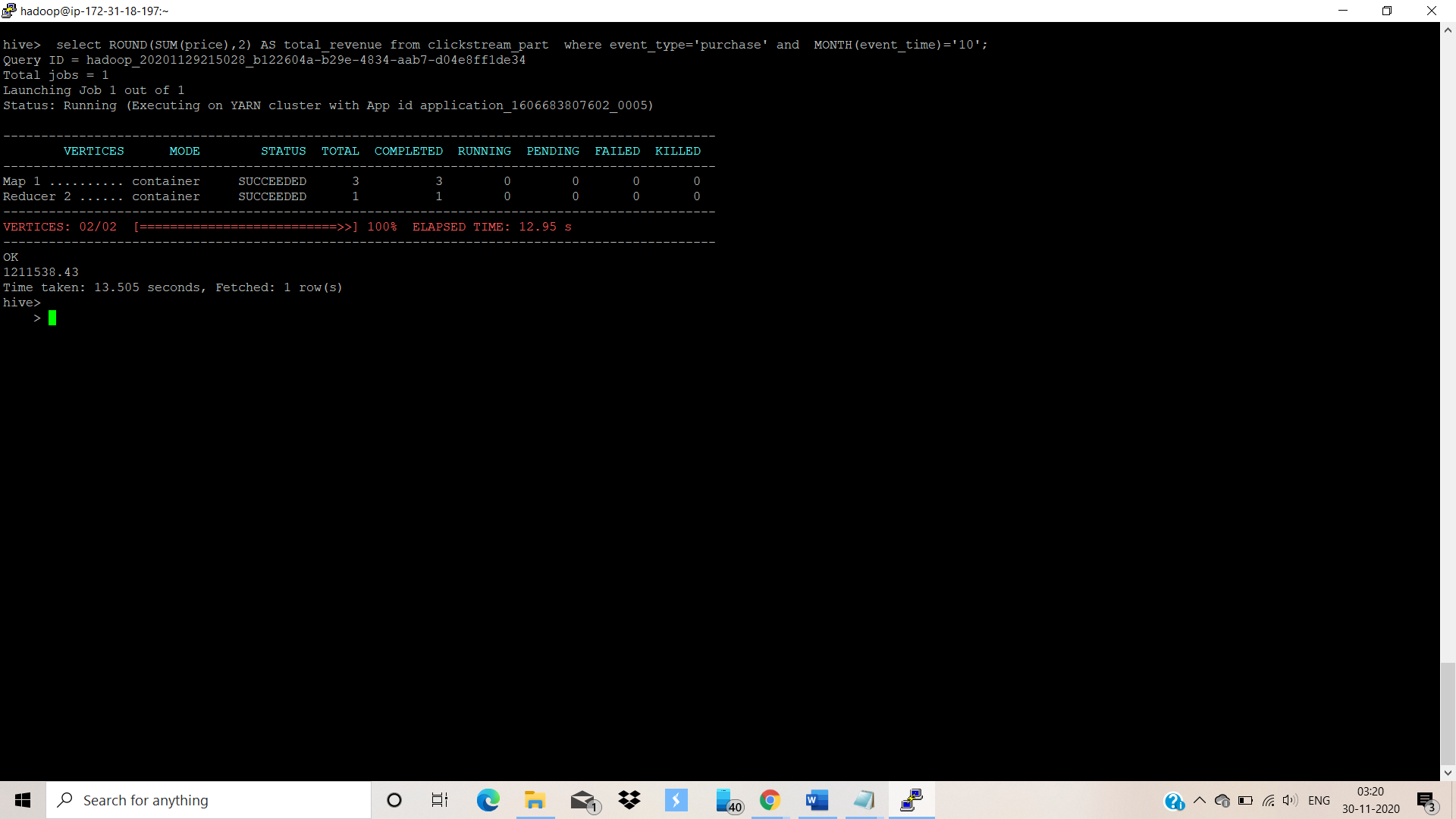
**Turning on the headers :**

set hive.cli.print.header=true ;

**Q1. Find the total revenue generated due to purchases made in October.**

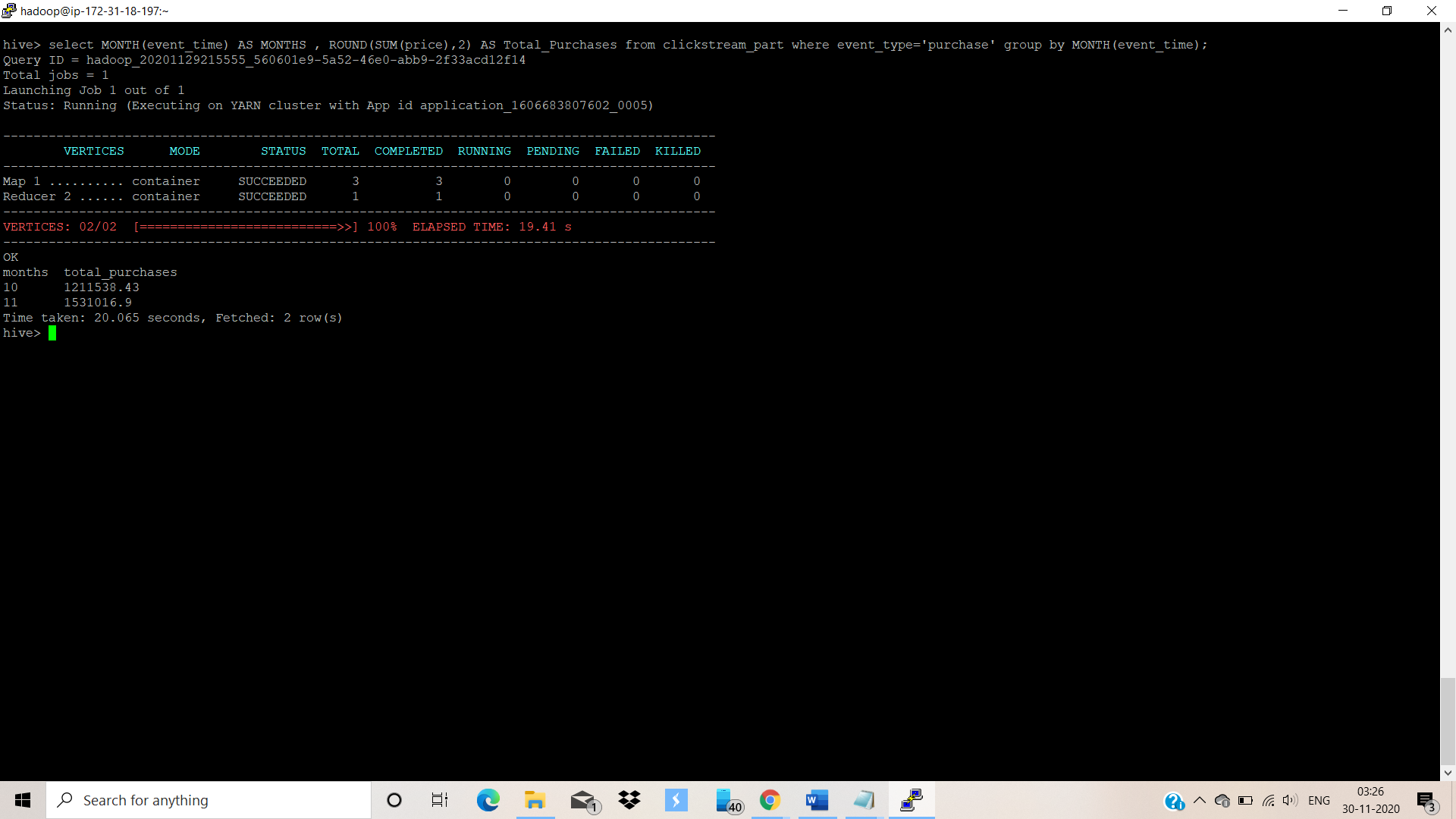
**Answer: 1211538.43**

Query : **hive> select ROUND(SUM(price),2) AS total\_revenue from clickstream\_part where event\_type='purchase' and MONTH(event\_time)='10';**



**Q2. Write a query to yield the total sum of purchases per month in a single output.**

hive> **select MONTH(event\_time) AS MONTHS , ROUND(SUM(price),2) AS Total\_Purchases from clickstream\_part where event\_type='purchase' group by MONTH(event\_time);**



**Q3. Write a query to find the change in revenue generated due to purchases from October to November.**

Query> **WITH month\_revenue AS**

**(SELECT**

**SUM(case when MONTH(event\_time) = '10' then price else 0 end) AS Oct\_Revenue,**

**SUM(case when MONTH(event\_time) = '11' then price else 0 end) AS Nov\_Revenue**

**FROM clickstream\_part**

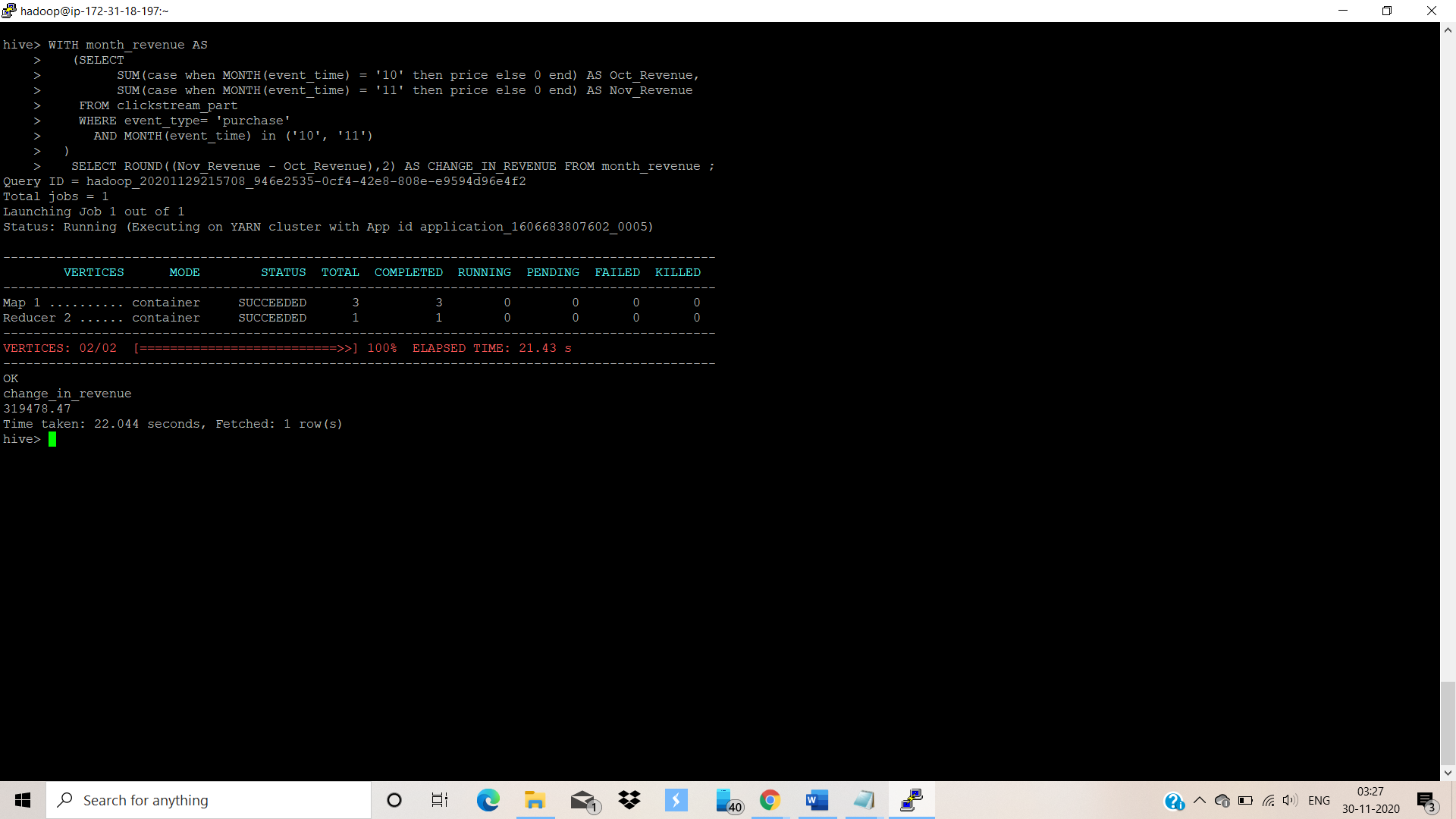
**WHERE event\_type= 'purchase'**

**AND MONTH(event\_time) in ('10', '11')**

**)**

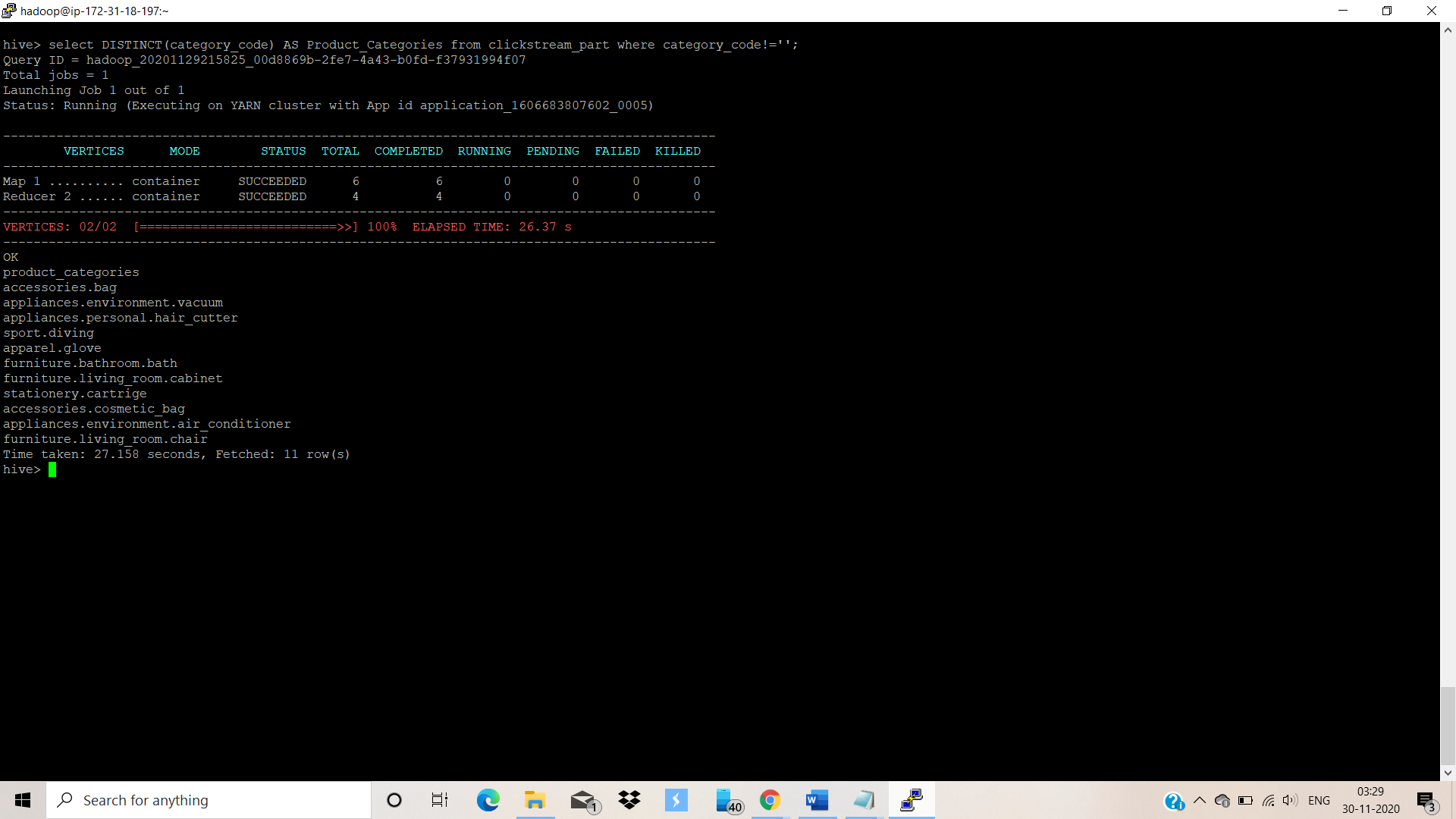
**SELECT ROUND((Nov\_Revenue - Oct\_Revenue),2) AS CHANGE\_IN\_REVENUE FROM month\_revenue ;**

**Answer : 319478.47**



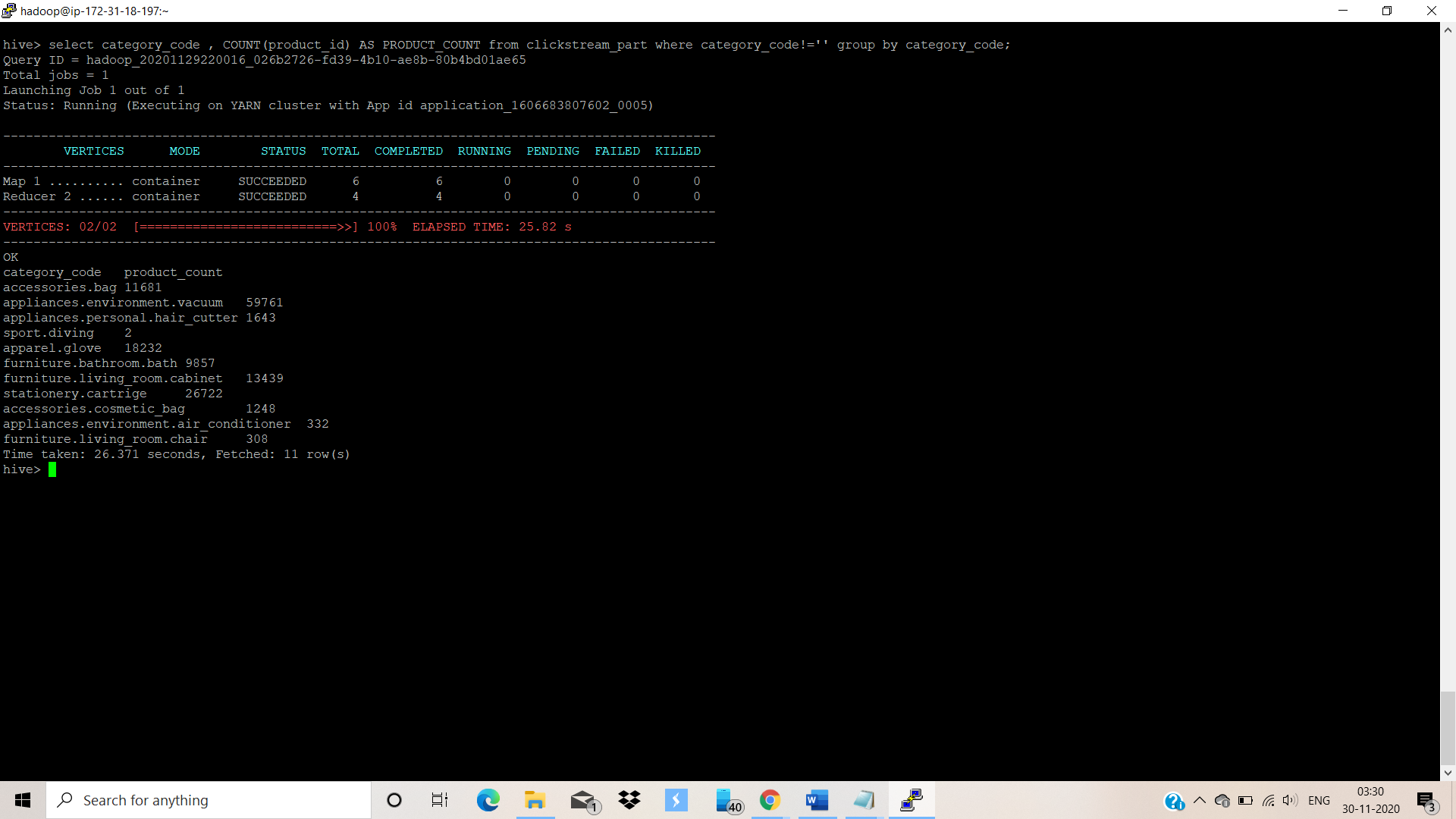
**Q4. Find distinct categories of products. Categories with null category code can be ignored**.

**Query> select DISTINCT(category\_code) AS Product\_Categories from clickstream\_part where category\_code!='';**



**Q5. Find the total number of products available under each category.**

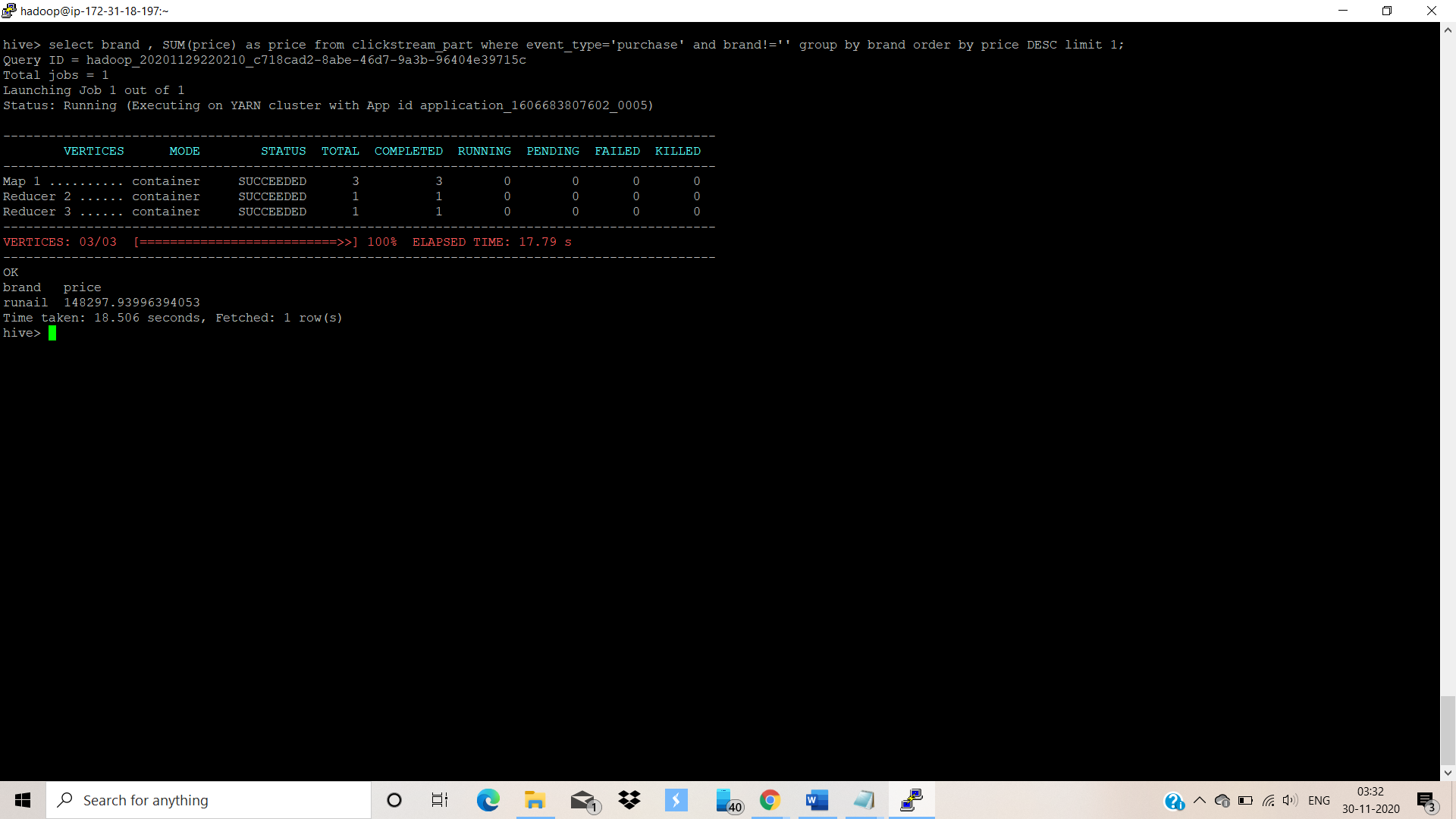
**Query> select category\_code , COUNT(product\_id) AS PRODUCT\_COUNT from clickstream\_part where category\_code!='' group by category\_code;**



**Q6. Which brand had the maximum sales in October and November**

**combined?**

**Query > select brand , SUM(price) as price from clickstream\_part where event\_type='purchase' and brand!='' group by brand order by price DESC limit 1;**



**Q7. Which brands increased their sales from October to November?**

Query🡪 WITH increased\_sales AS(

select brand, SUM(case when MONTH(event\_time) = '10' then price else 0 end) AS Oct\_Revenue,

SUM(case when MONTH(event\_time) = '11' then price else 0 end) AS Nov\_Revenue

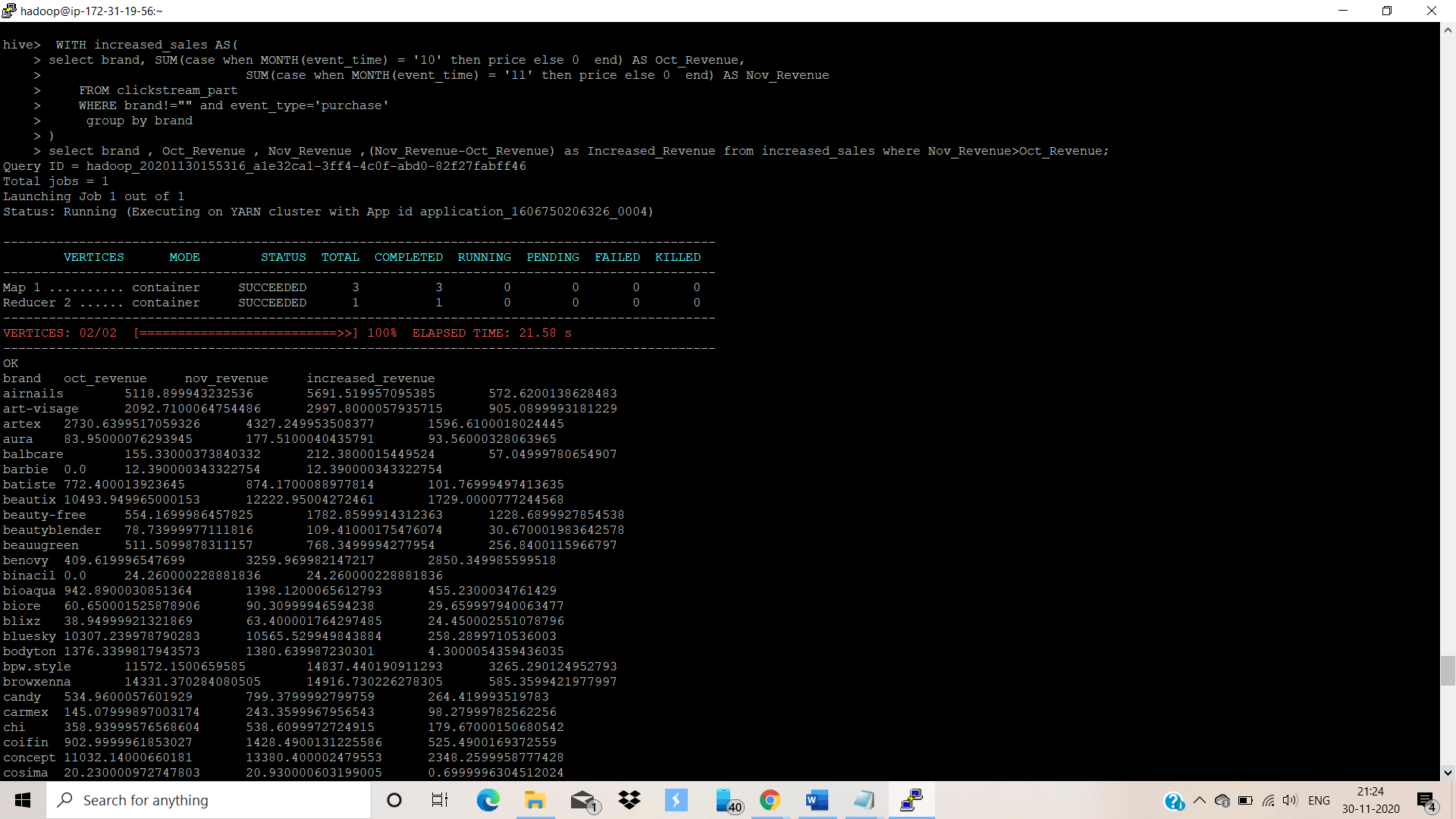
FROM clickstream\_part

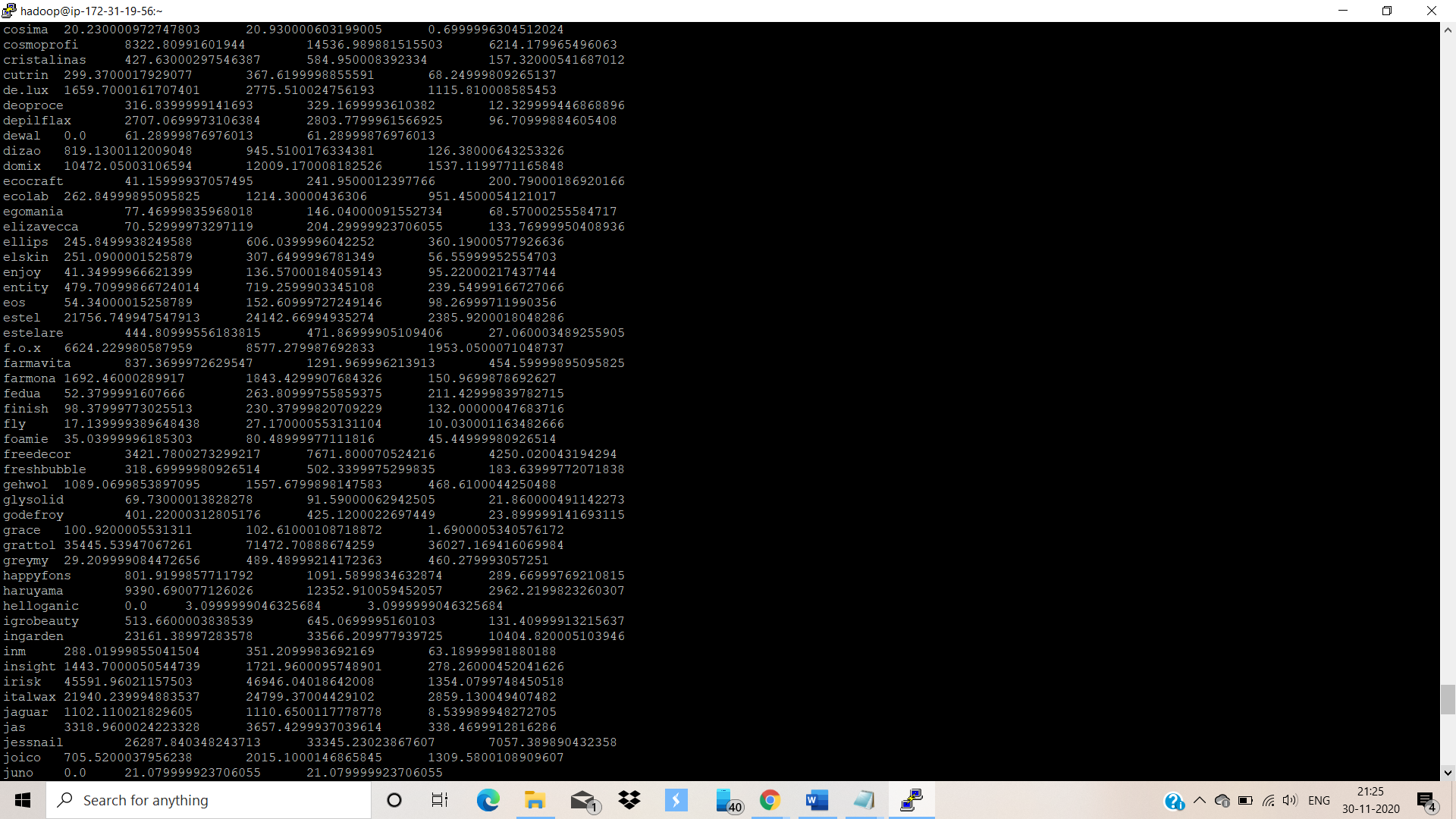
WHERE brand!="" and event\_type='purchase'

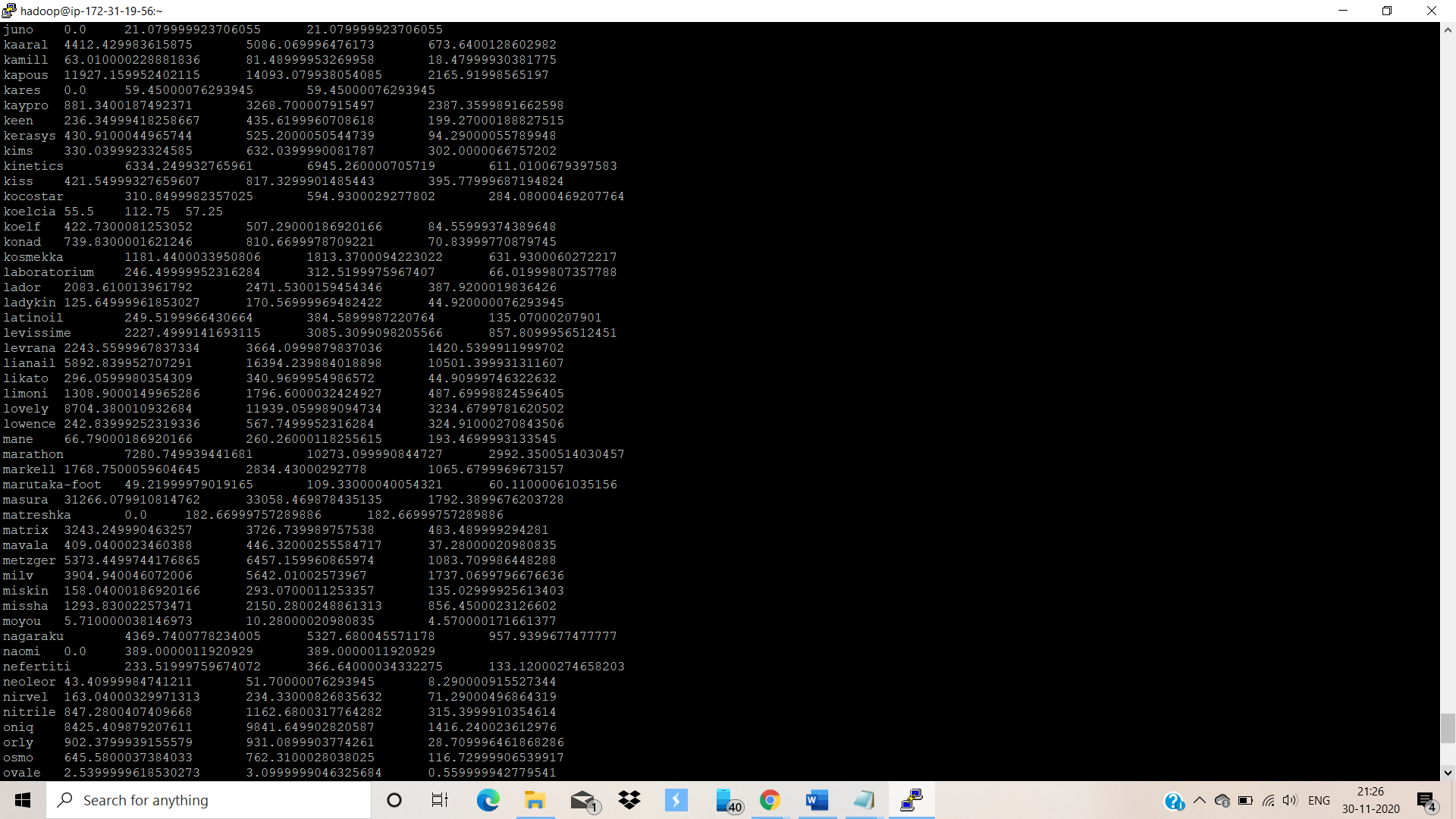
group by brand

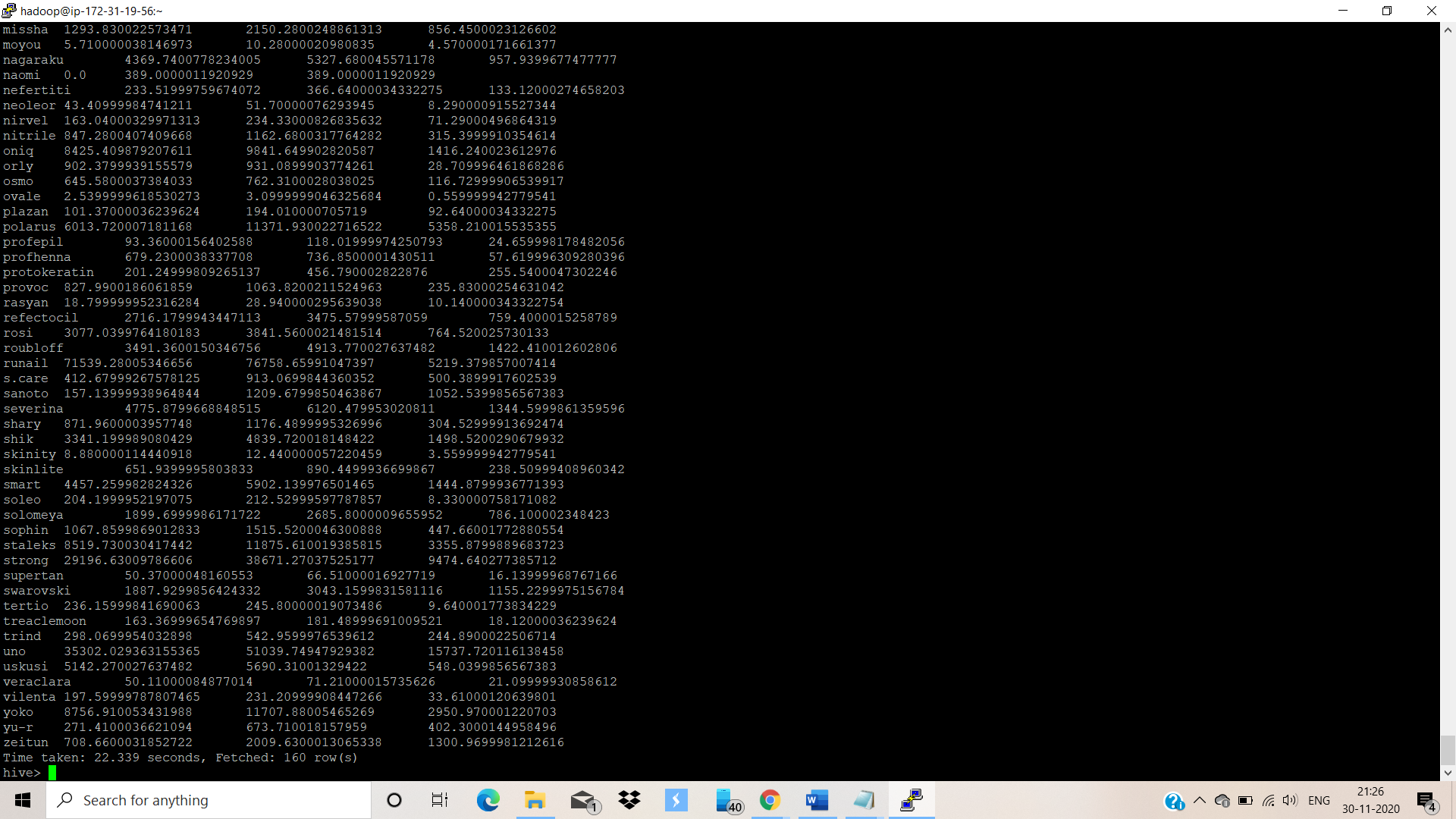
)

select brand , Oct\_Revenue , Nov\_Revenue ,(Nov\_Revenue-Oct\_Revenue) as Increased\_Revenue from increased\_sales where Nov\_Revenue>Oct\_Revenue;





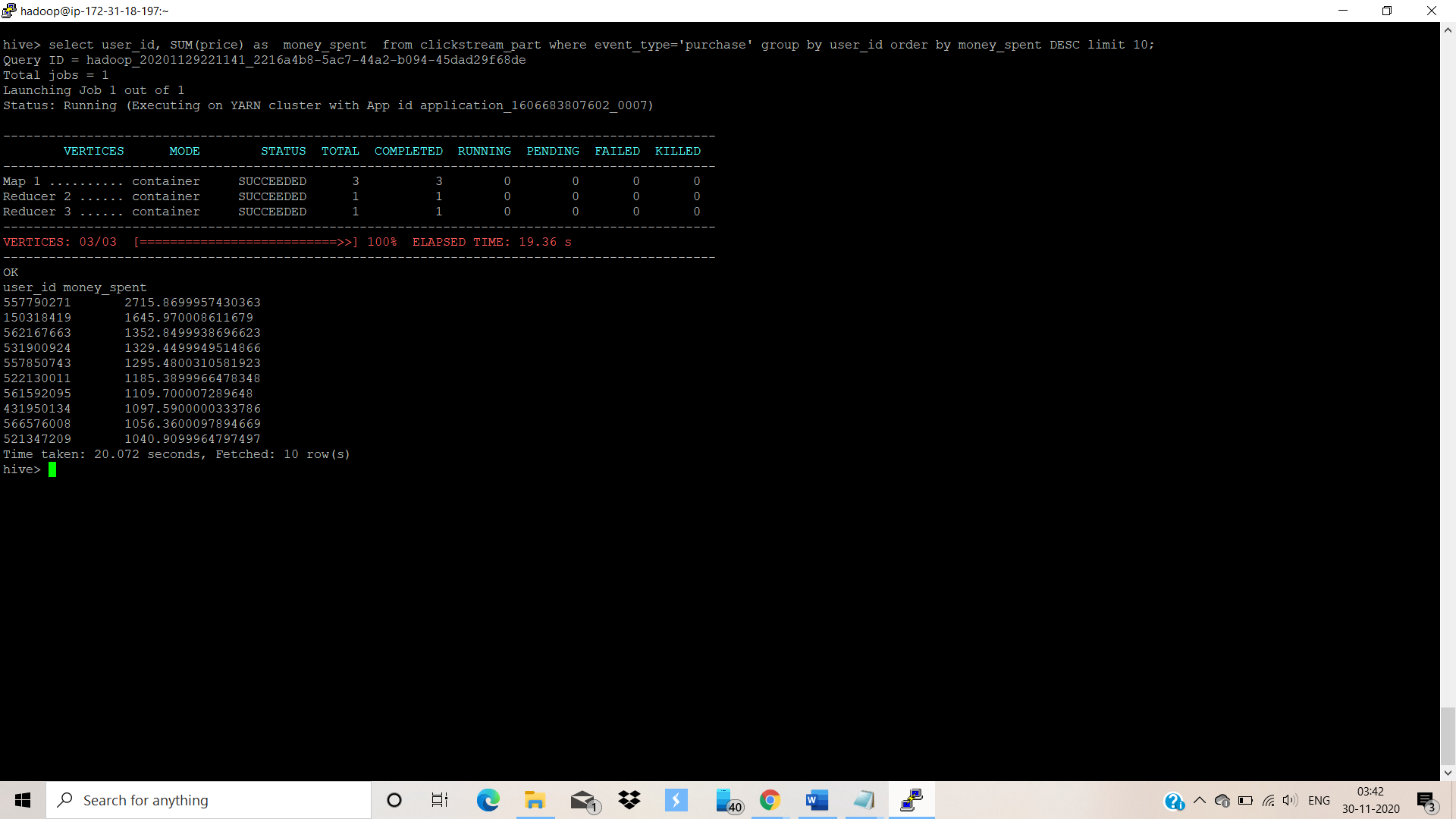




**Q8. Your company wants to reward the top 10 users of its website with a Golden Customer plan. Write a query to generate a list of top 10 users who spend the most.**

Query-> select user\_id , SUM(price) as money\_spent from clickstream\_part where event\_type=

'purchase' group by user\_id order by money\_spent DESC limit 10 ;



**Dropping the database and terminating the cluster :**

