INFO 7250 – ENGINEERING OF BIG DATA SYSTEMS
SEC 01

ANALYSING IOWA LIQUOR SALES USING HADOOP

GAURESH CHAVAN

001854530

INDEX

INTRODUCTION

Problem Statement:

The objective of this project is to explore the dataset and implement Hadoop ecosystem components on it. We are primarily going to utilize Hadoop's MapReduce , Pig and Hive to derive some analytic insights from data.

Data Glossary:

This dataset contains the spirits purchase information of Iowa Class "E" liquor licenses by product and date of purchase from January 1, 2012 to current. The dataset can be used to analyze total spirits sales in Iowa of individual products at the store level.

Updated: November 2, 2019

Data provided by: Iowa Department of Commerce, Alcoholic Beverages Division

This dataset has 17.3 Million rows across 17 columns. Each row represents a transaction.

- Invoice/Item Number
- Date
- Store Number
- Store Name
- Address
- City
- Zip Code
- Store Location
- County Number
- County
- Category
- Category Name
- Vendor Number
- Vendor Name
- Item Number
- Item Description
- Pack
- Bottle Volume (ml)
- State Bottle Cost
- State Bottle Retail
- Bottles Sold
- Sale (Dollars)
- Volume Sold (Liters)
- Volume Sold (Gallons

TECHNOLOGY

Apache MapReduce:

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.

Code for MapReduce is written in Java. Eclipse IDE is used as a development environment.

Implemented on Use Case 3.1 & 3.2

Apache Pig:

Apache Pig is a high-level platform for creating programs that run on Apache Hadoop. The language for this platform is called Pig Latin. Pig can execute its Hadoop jobs in MapReduce, Apache Tez, or Apache Spark.

Implemented on Use Case 3.3 & 3.4

Apache Hive:

Apache Hive is a data warehouse software project built on top of Apache Hadoop for providing data query and analysis. Hive gives a SQL-like interface to query data stored in various databases and file systems that integrate with Hadoop.

Implemented on Use Case 3.5

Both Apache Pig & Hive is being used owing to its fast processing and slow learning curve. While both are languages (Pig is a data flow language, Hive provides SQL constructs), there is a little learning curve and reduces the number of lines of code mapper/reducer would otherwise take.

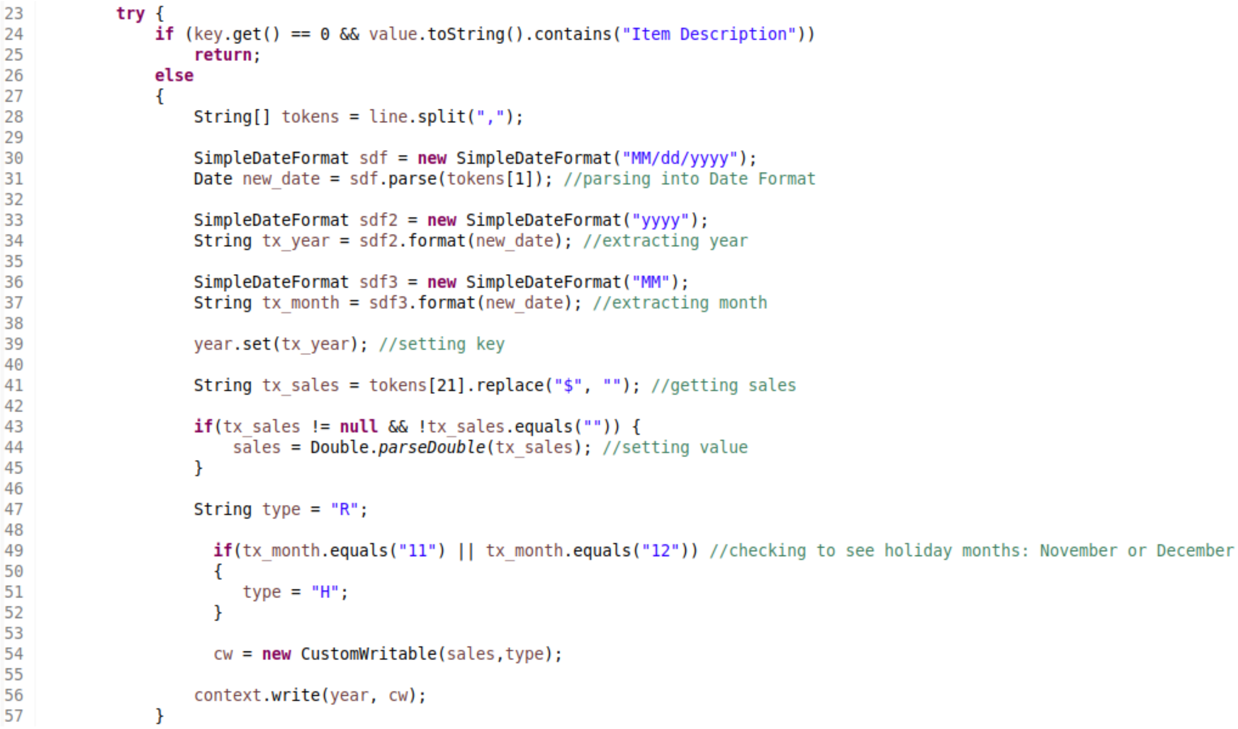
USE CASES

1. Analyzing liquor sales during Holiday Season vs Regular Season

It would not be surprising to assume, people naturally consume more alcohol during holiday season i.e. owing to Thanksgiving, Hanukkah, Christmas and New Year's. So, it would be interesting to see if our assumption is correct. We will be comparing sales for first 10 months i.e. January – October to those for last 2 months i.e. November - December for every year from 2012- 2019.

Our comparison metrics will be **Average Monthly Sales** & **Average sale per transaction** for that year

Mapper:



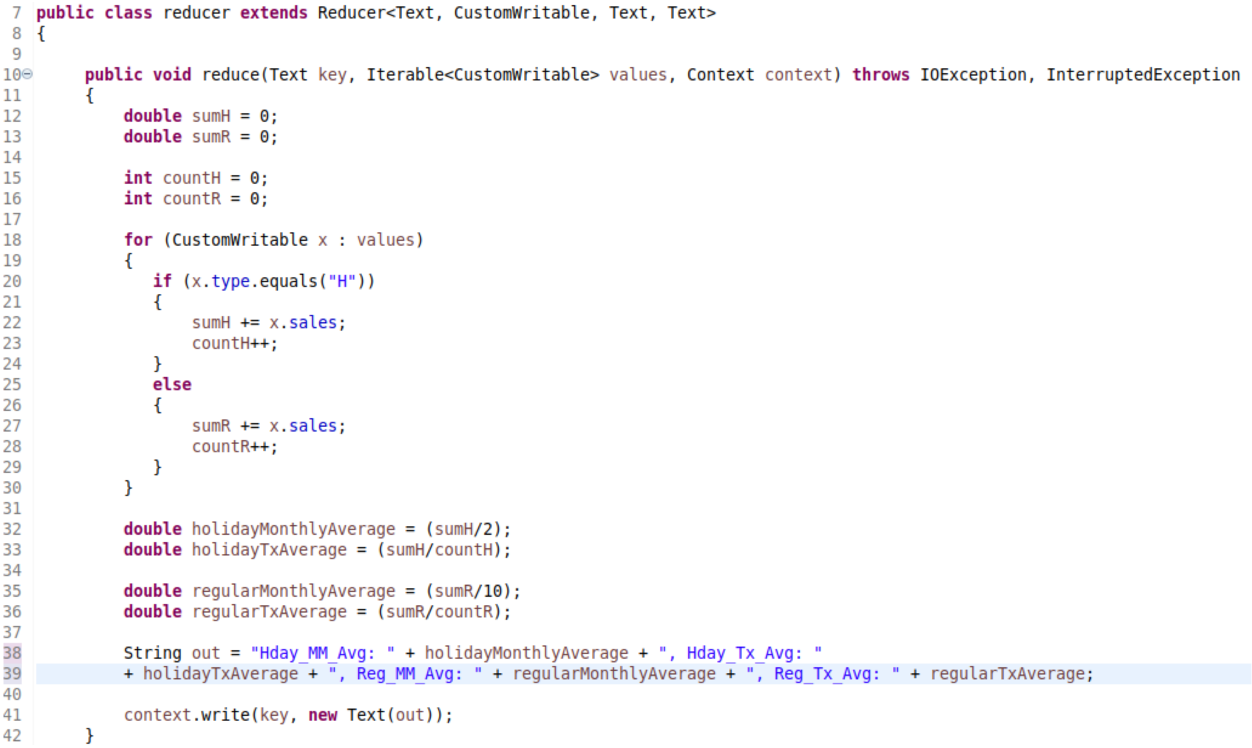
```
        try {
            if (key.get() == 0 && value.toString().contains("Item Description"))
                return;
            else
            {
                String[] tokens = line.split(",");

                SimpleDateFormat sdf = new SimpleDateFormat("MM/dd/yyyy");
                Date new_date = sdf.parse(tokens[1]); //parsing into Date Format

                SimpleDateFormat sdf2 = new SimpleDateFormat("yyyy");
                String tx_year = sdf2.format(new_date); //extracting year

                SimpleDateFormat sdf3 = new SimpleDateFormat("MM");
                String tx_month = sdf3.format(new_date); //extracting month

                year.set(tx_year); //setting key

                String tx_sales = tokens[21].replace("$", ""); //getting sales

                if(tx_sales != null && !tx_sales.equals("")) {
                    sales = Double.parseDouble(tx_sales); //setting value
                }

                String type = "R";

                  if(tx_month.equals("11") || tx_month.equals("12")) //checking to see holiday months: November or December
                  {
                     type = "H";
                  }

                  cw = new CustomWritable(sales,type);

                context.write(year, cw);
            }
```

Since the date we have in our data is a string, we parse it into *Date()* format. Then we extract 'Year' which is our key, and 'month' which will be used to grab key-value pairs as 'Holiday' sales. Also, we define **a type == 'H', 'R'** which acts as a filter to separate the sales when we reduce them.

Reducer:

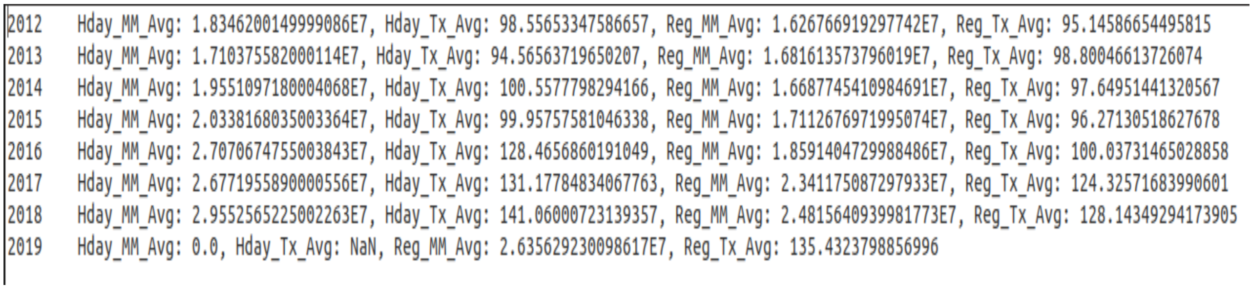
In reducer, we define two separate counts and sums; one of holiday and other for regular average. If type = H, we calculate monthly average as (sum/2) i.e. for November &

December, else we calculate regular monthly average as (sum/10) i.e. for January – October.



```java
public class reducer extends Reducer<Text, CustomWritable, Text, Text>
{

    public void reduce(Text key, Iterable<CustomWritable> values, Context context) throws IOException, InterruptedException
    {
        double sumH = 0;
        double sumR = 0;

        int countH = 0;
        int countR = 0;

        for (CustomWritable x : values)
        {
           if (x.type.equals("H"))
           {
               sumH += x.sales;
               countH++;
           }
           else
           {
               sumR += x.sales;
               countR++;
           }
        }

        double holidayMonthlyAverage = (sumH/2);
        double holidayTxAverage = (sumH/countH);

        double regularMonthlyAverage = (sumR/10);
        double regularTxAverage = (sumR/countR);

        String out = "Hday_MM_Avg: " + holidayMonthlyAverage + ", Hday_Tx_Avg: "
        + holidayTxAverage + ", Reg_MM_Avg: " + regularMonthlyAverage + ", Reg_Tx_Avg: " + regularTxAverage;

        context.write(key, new Text(out));
    }
```

Output:



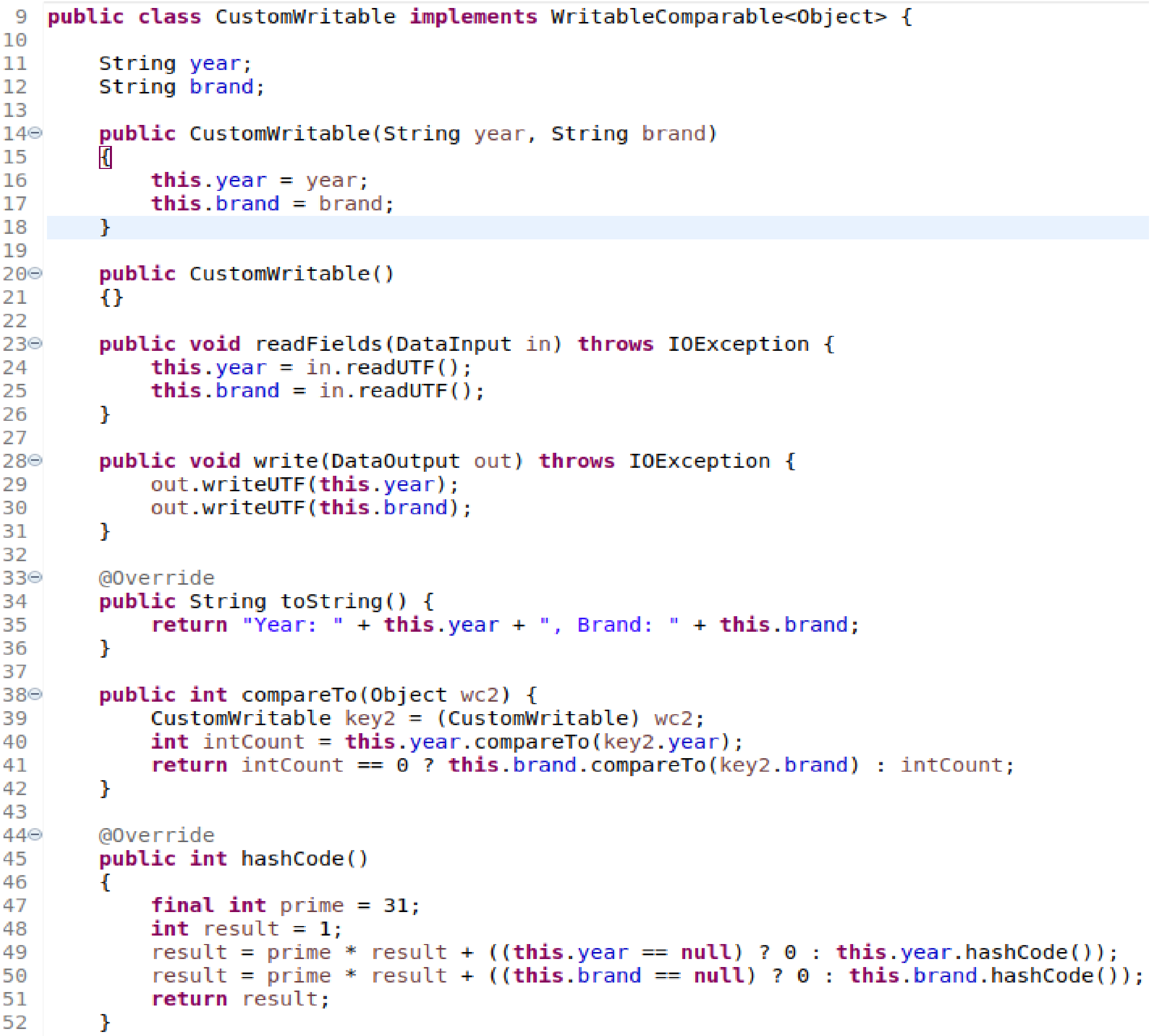
```
2012    Hday_MM_Avg: 1.8346200149999086E7, Hday_Tx_Avg: 98.55653347586657, Reg_MM_Avg: 1.626766919297742E7, Reg_Tx_Avg: 95.14586654495815
2013    Hday_MM_Avg: 1.710375582000114E7, Hday_Tx_Avg: 94.56563719650207, Reg_MM_Avg: 1.681613573796019E7, Reg_Tx_Avg: 98.80046613726074
2014    Hday_MM_Avg: 1.9551097180004068E7, Hday_Tx_Avg: 100.5577798294166, Reg_MM_Avg: 1.6687745410984691E7, Reg_Tx_Avg: 97.64951441320567
2015    Hday_MM_Avg: 2.0338168035003364E7, Hday_Tx_Avg: 99.95757581046338, Reg_MM_Avg: 1.7112676971995074E7, Reg_Tx_Avg: 96.27130518627678
2016    Hday_MM_Avg: 2.7070674755003843E7, Hday_Tx_Avg: 128.4656860191049, Reg_MM_Avg: 1.8591404729988486E7, Reg_Tx_Avg: 100.03731465028858
2017    Hday_MM_Avg: 2.6771955890000556E7, Hday_Tx_Avg: 131.17784834067763, Reg_MM_Avg: 2.341175087297933E7, Reg_Tx_Avg: 124.32571683990601
2018    Hday_MM_Avg: 2.9552565225002263E7, Hday_Tx_Avg: 141.06000723139357, Reg_MM_Avg: 2.4815640939981773E7, Reg_Tx_Avg: 128.14349294173905
2019    Hday_MM_Avg: 0.0, Hday_Tx_Avg: NaN, Reg_MM_Avg: 2.635629230098617E7, Reg_Tx_Avg: 135.4323798856996
```

2. Assessing top 5 liquor labels based on their sales from 2012-2019

WritableComparable:

To calculate top 5 liquor brands per year by sales, we create a CustomWritable class where we define a composite key (year, brand) with value being (sales).

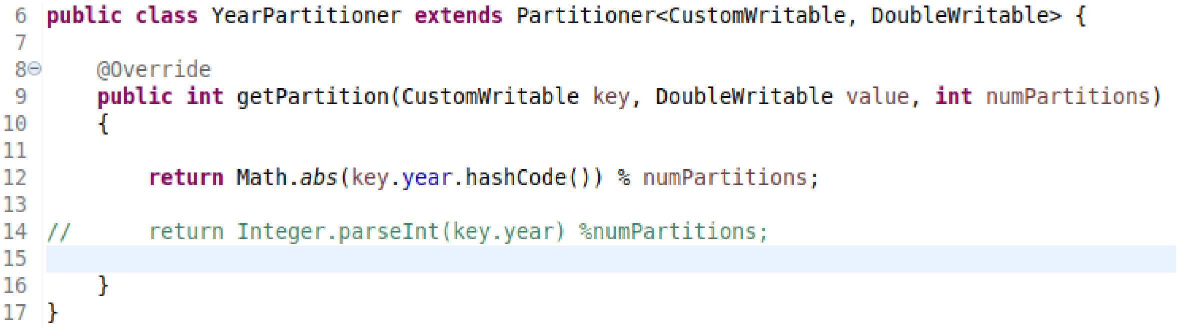
The compareTo() is WritableComparable's natural comparison method. It returns 0 if both objects are same and with a number higher or lower you determine the order between your objects. In our case, we compare our composite key elements: year, brand.



```java
public class CustomWritable implements WritableComparable<Object> {

    String year;
    String brand;

    public CustomWritable(String year, String brand)
    {
        this.year = year;
        this.brand = brand;
    }

    public CustomWritable()
    {}

    public void readFields(DataInput in) throws IOException {
        this.year = in.readUTF();
        this.brand = in.readUTF();
    }

    public void write(DataOutput out) throws IOException {
        out.writeUTF(this.year);
        out.writeUTF(this.brand);
    }

    @Override
    public String toString() {
        return "Year: " + this.year + ", Brand: " + this.brand;
    }

    public int compareTo(Object wc2) {
        CustomWritable key2 = (CustomWritable) wc2;
        int intCount = this.year.compareTo(key2.year);
        return intCount == 0 ? this.brand.compareTo(key2.brand) : intCount;
    }

    @Override
    public int hashCode()
    {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((this.year == null) ? 0 : this.year.hashCode());
        result = prime * result + ((this.brand == null) ? 0 : this.brand.hashCode());
        return result;
    }
```

Partitioner:

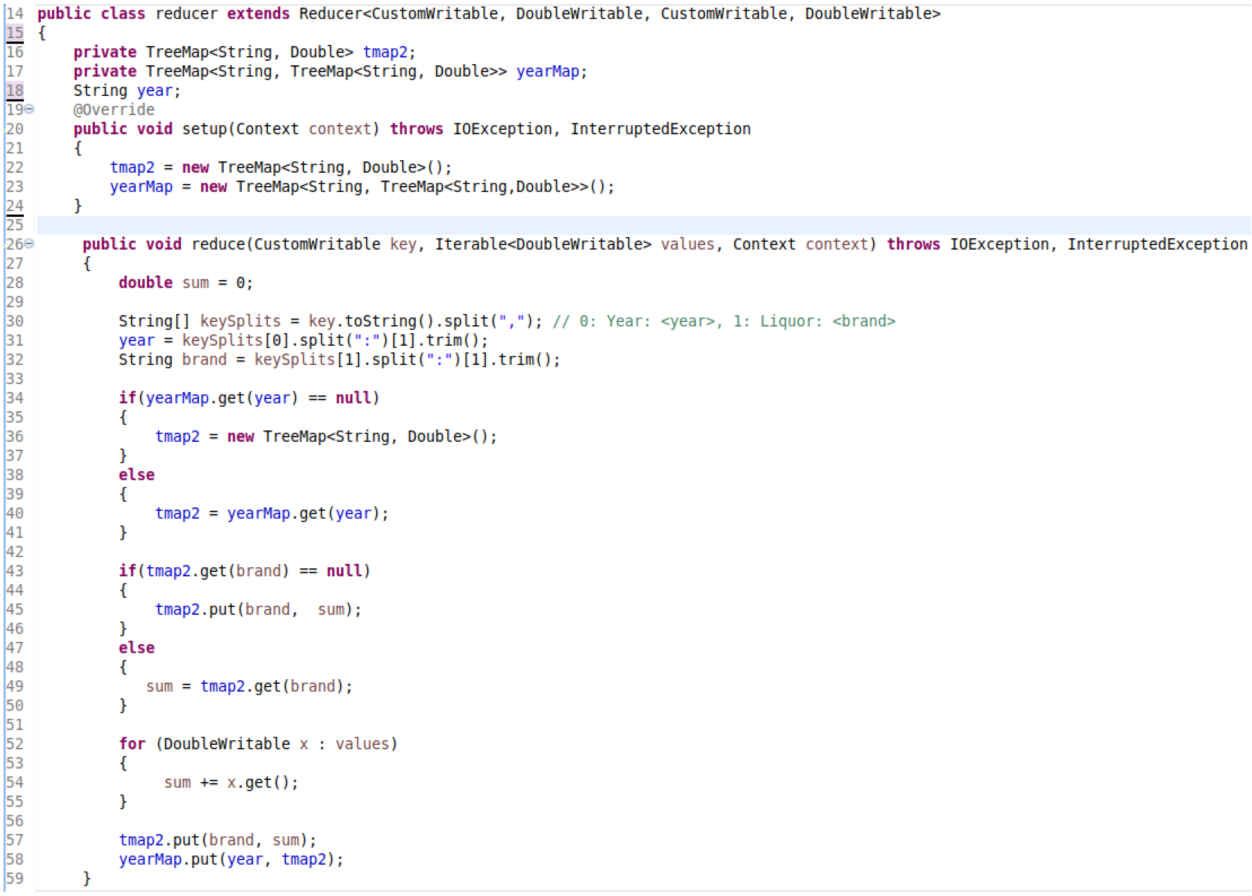
By default, the partitioner implementation is called HashPartitioner. It uses the hashCode() method of the key objects modulo the number of partitions total to determine which partition to send a given (key, value) pair to. The getPartition() method receives a key and a value and the number of partitions to split the data, a number in the range [0, numPartitions) must be returned by this method, indicating which partition to send the key and value to.



```java
public class YearPartitioner extends Partitioner<CustomWritable, DoubleWritable> {

    @Override
    public int getPartition(CustomWritable key, DoubleWritable value, int numPartitions)
    {

        return Math.abs(key.year.hashCode()) % numPartitions;

//      return Integer.parseInt(key.year) %numPartitions;

    }
}
```

In our reducer, we define two tree maps: one to store count of sales of all the brands sold in a given year; other to store the first map per year i.e. for first tree map, brand is the key whereas for second map, year will behave as a key.

Reducer:

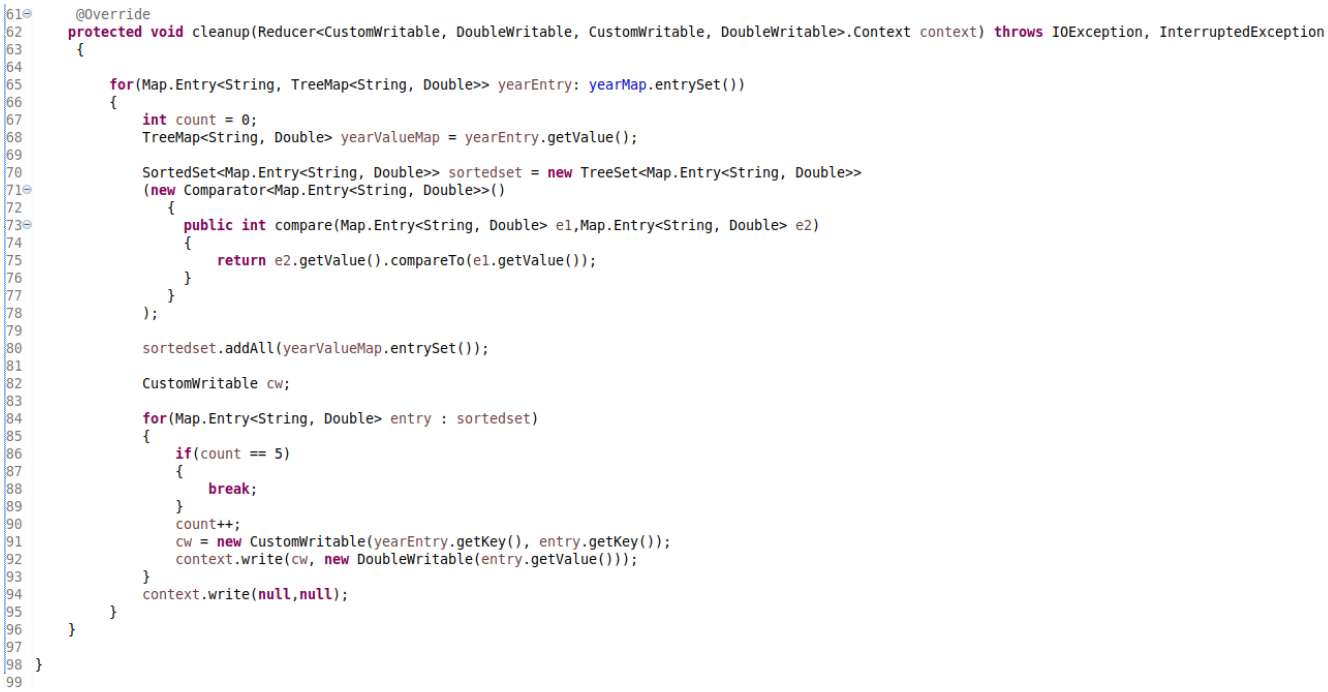


```
public class reducer extends Reducer<CustomWritable, DoubleWritable, CustomWritable, DoubleWritable>
{
    private TreeMap<String, Double> tmap2;
    private TreeMap<String, TreeMap<String, Double>> yearMap;
    String year;
    @Override
    public void setup(Context context) throws IOException, InterruptedException
    {
        tmap2 = new TreeMap<String, Double>();
        yearMap = new TreeMap<String, TreeMap<String,Double>>();
    }

     public void reduce(CustomWritable key, Iterable<DoubleWritable> values, Context context) throws IOException, InterruptedException
     {
         double sum = 0;

         String[] keySplits = key.toString().split(","); // 0: Year: <year>, 1: Liquor: <brand>
         year = keySplits[0].split(":")[1].trim();
         String brand = keySplits[1].split(":")[1].trim();

         if(yearMap.get(year) == null)
         {
             tmap2 = new TreeMap<String, Double>();
         }
         else
         {
             tmap2 = yearMap.get(year);
         }

         if(tmap2.get(brand) == null)
         {
             tmap2.put(brand,  sum);
         }
         else
         {
            sum = tmap2.get(brand);
         }

         for (DoubleWritable x : values)
         {
              sum += x.get();
         }

         tmap2.put(brand, sum);
         yearMap.put(year, tmap2);
     }
```

Cleanup:

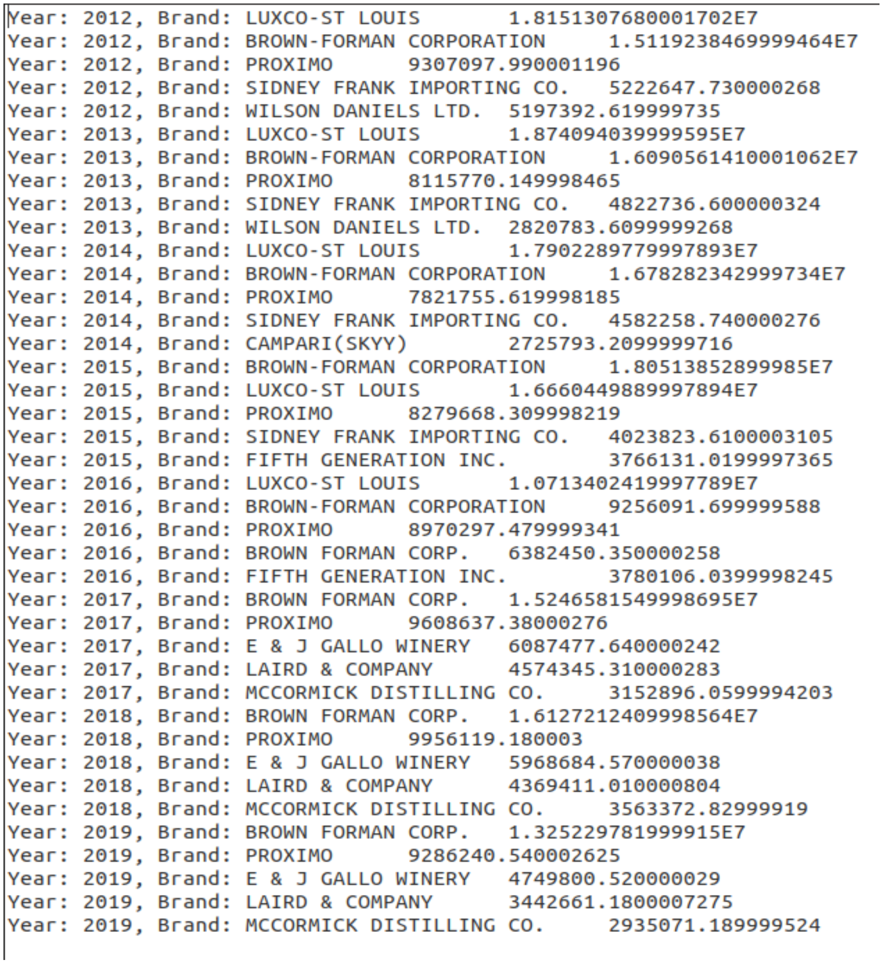
The first step in clean up is to sort the values based on sales (value) coming from year map which is why we see it as an entry set to a Sorted Set. The **Java Sorted Set** interface behaves like a normal **Set** with the exception that the elements it contains are **sorted** internally. This means that when you iterate the elements of a **Sorted Set** the elements are iterated in the **sorted** order.

In a way sorted set assumes sorting thus relieving us of using a sort comparator. Second, we initialize a count = 0 which is then used to iterate over the sorted set to print out only 5 top values which is our desired result.



```java
     @Override
    protected void cleanup(Reducer<CustomWritable, DoubleWritable, CustomWritable, DoubleWritable>.Context context) throws IOException, InterruptedException
     {

         for(Map.Entry<String, TreeMap<String, Double>> yearEntry: yearMap.entrySet())
         {
             int count = 0;
             TreeMap<String, Double> yearValueMap = yearEntry.getValue();

             SortedSet<Map.Entry<String, Double>> sortedset = new TreeSet<Map.Entry<String, Double>>
             (new Comparator<Map.Entry<String, Double>>()
                {
                  public int compare(Map.Entry<String, Double> e1,Map.Entry<String, Double> e2)
                  {
                      return e2.getValue().compareTo(e1.getValue());
                  }
                }
             );

             sortedset.addAll(yearValueMap.entrySet());

             CustomWritable cw;

             for(Map.Entry<String, Double> entry : sortedset)
             {
                 if(count == 5)
                 {
                     break;
                 }
                 count++;
                 cw = new CustomWritable(yearEntry.getKey(), entry.getKey());
                 context.write(cw, new DoubleWritable(entry.getValue()));
             }
             context.write(null,null);
         }
    }

}
```

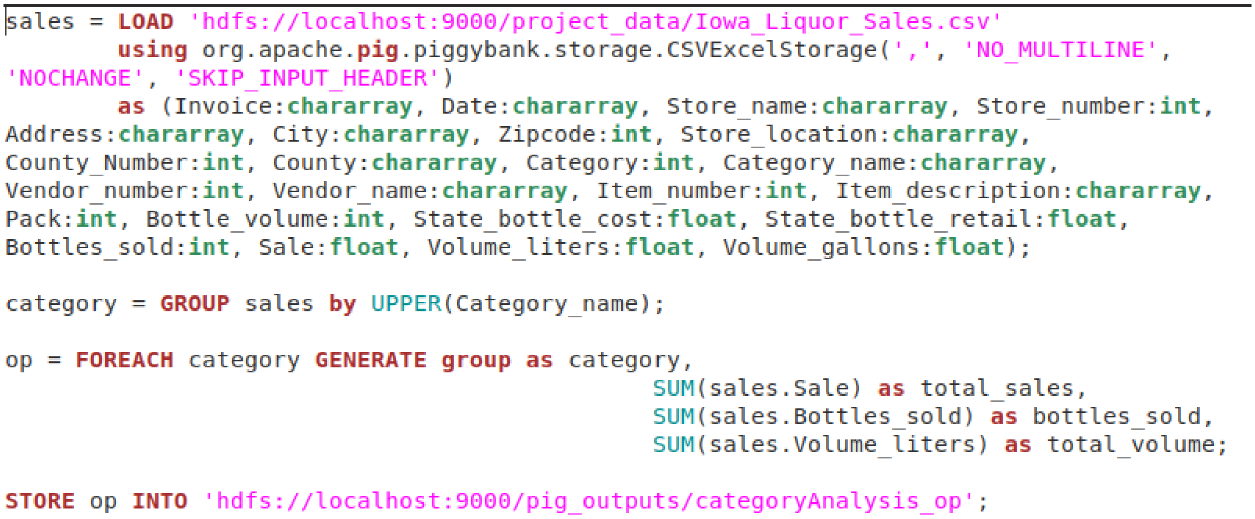
Output:



```
Year: 2012, Brand: LUXCO-ST LOUIS       1.8151307680001702E7
Year: 2012, Brand: BROWN-FORMAN CORPORATION     1.5119238469999464E7
Year: 2012, Brand: PROXIMO      9307097.990001196
Year: 2012, Brand: SIDNEY FRANK IMPORTING CO.   5222647.730000268
Year: 2012, Brand: WILSON DANIELS LTD.  5197392.619999735
Year: 2013, Brand: LUXCO-ST LOUIS       1.874094039999595E7
Year: 2013, Brand: BROWN-FORMAN CORPORATION     1.6090561410001062E7
Year: 2013, Brand: PROXIMO      8115770.149998465
Year: 2013, Brand: SIDNEY FRANK IMPORTING CO.   4822736.600000324
Year: 2013, Brand: WILSON DANIELS LTD.  2820783.6099999268
Year: 2014, Brand: LUXCO-ST LOUIS       1.7902289779997893E7
Year: 2014, Brand: BROWN-FORMAN CORPORATION     1.678282342999734E7
Year: 2014, Brand: PROXIMO      7821755.619998185
Year: 2014, Brand: SIDNEY FRANK IMPORTING CO.   4582258.740000276
Year: 2014, Brand: CAMPARI(SKYY)        2725793.2099999716
Year: 2015, Brand: BROWN-FORMAN CORPORATION     1.80513852899985E7
Year: 2015, Brand: LUXCO-ST LOUIS       1.6660449889997894E7
Year: 2015, Brand: PROXIMO      8279668.309998219
Year: 2015, Brand: SIDNEY FRANK IMPORTING CO.   4023823.6100003105
Year: 2015, Brand: FIFTH GENERATION INC.        3766131.0199997365
Year: 2016, Brand: LUXCO-ST LOUIS       1.0713402419997789E7
Year: 2016, Brand: BROWN-FORMAN CORPORATION     9256091.699999588
Year: 2016, Brand: PROXIMO      8970297.479999341
Year: 2016, Brand: BROWN FORMAN CORP.   6382450.350000258
Year: 2016, Brand: FIFTH GENERATION INC.        3780106.0399998245
Year: 2017, Brand: BROWN FORMAN CORP.   1.5246581549998695E7
Year: 2017, Brand: PROXIMO      9608637.38000276
Year: 2017, Brand: E & J GALLO WINERY   6087477.640000242
Year: 2017, Brand: LAIRD & COMPANY      4574345.310000283
Year: 2017, Brand: MCCORMICK DISTILLING CO.     3152896.0599994203
Year: 2018, Brand: BROWN FORMAN CORP.   1.6127212409998564E7
Year: 2018, Brand: PROXIMO      9956119.180003
Year: 2018, Brand: E & J GALLO WINERY   5968684.570000038
Year: 2018, Brand: LAIRD & COMPANY      4369411.010000804
Year: 2018, Brand: MCCORMICK DISTILLING CO.     3563372.82999919
Year: 2019, Brand: BROWN FORMAN CORP.   1.325229781999915E7
Year: 2019, Brand: PROXIMO      9286240.540002625
Year: 2019, Brand: E & J GALLO WINERY   4749800.520000029
Year: 2019, Brand: LAIRD & COMPANY      3442661.1800007275
Year: 2019, Brand: MCCORMICK DISTILLING CO.     2935071.189999524
```

3. Investigating alcohol category performance over the years

Pig Script:

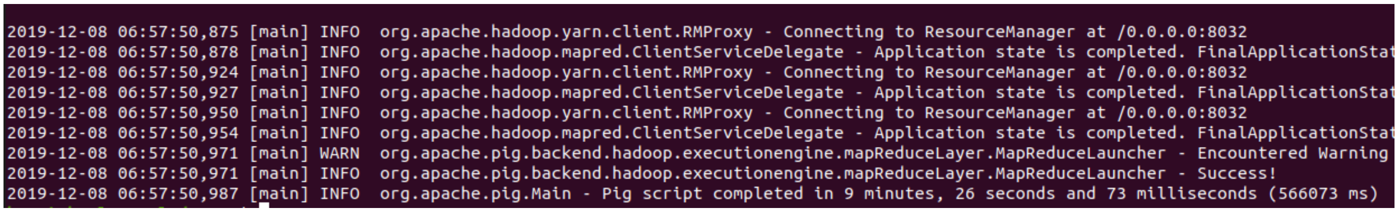


```
sales = LOAD 'hdfs://localhost:9000/project_data/Iowa_Liquor_Sales.csv'
        using org.apache.pig.piggybank.storage.CSVExcelStorage(',', 'NO_MULTILINE',
'NOCHANGE', 'SKIP_INPUT_HEADER')
        as (Invoice:chararray, Date:chararray, Store_name:chararray, Store_number:int,
Address:chararray, City:chararray, Zipcode:int, Store_location:chararray,
County_Number:int, County:chararray, Category:int, Category_name:chararray,
Vendor_number:int, Vendor_name:chararray, Item_number:int, Item_description:chararray,
Pack:int, Bottle_volume:int, State_bottle_cost:float, State_bottle_retail:float,
Bottles_sold:int, Sale:float, Volume_liters:float, Volume_gallons:float);

category = GROUP sales by UPPER(Category_name);

op = FOREACH category GENERATE group as category,
                                              SUM(sales.Sale) as total_sales,
                                              SUM(sales.Bottles_sold) as bottles_sold,
                                              SUM(sales.Volume_liters) as total_volume;

STORE op INTO 'hdfs://localhost:9000/pig_outputs/categoryAnalysis_op';
```

This is a straightforward script.

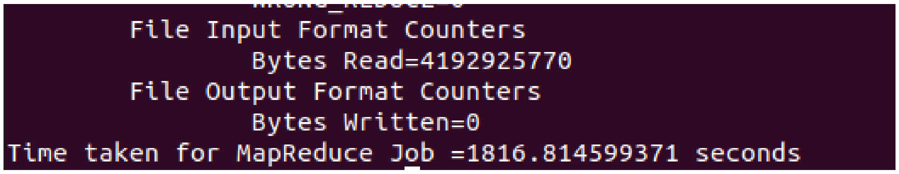
1. Load data into 'sales' (while skipping the header row)
2. Group by Category name
3. For each category name, compute total sales, total bottles sold, and total volume sold
4. Store output to HDFS

Pig Runtime:



```
2019-12-08 06:57:50,875 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-12-08 06:57:50,878 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStat
2019-12-08 06:57:50,924 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-12-08 06:57:50,927 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStat
2019-12-08 06:57:50,950 [main] INFO  org.apache.hadoop.yarn.client.RMProxy - Connecting to ResourceManager at /0.0.0.0:8032
2019-12-08 06:57:50,954 [main] INFO  org.apache.hadoop.mapred.ClientServiceDelegate - Application state is completed. FinalApplicationStat
2019-12-08 06:57:50,971 [main] WARN  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Encountered Warning
2019-12-08 06:57:50,971 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.MapReduceLauncher - Success!
2019-12-08 06:57:50,987 [main] INFO  org.apache.pig.Main - Pig script completed in 9 minutes, 26 seconds and 73 milliseconds (566073 ms)
```

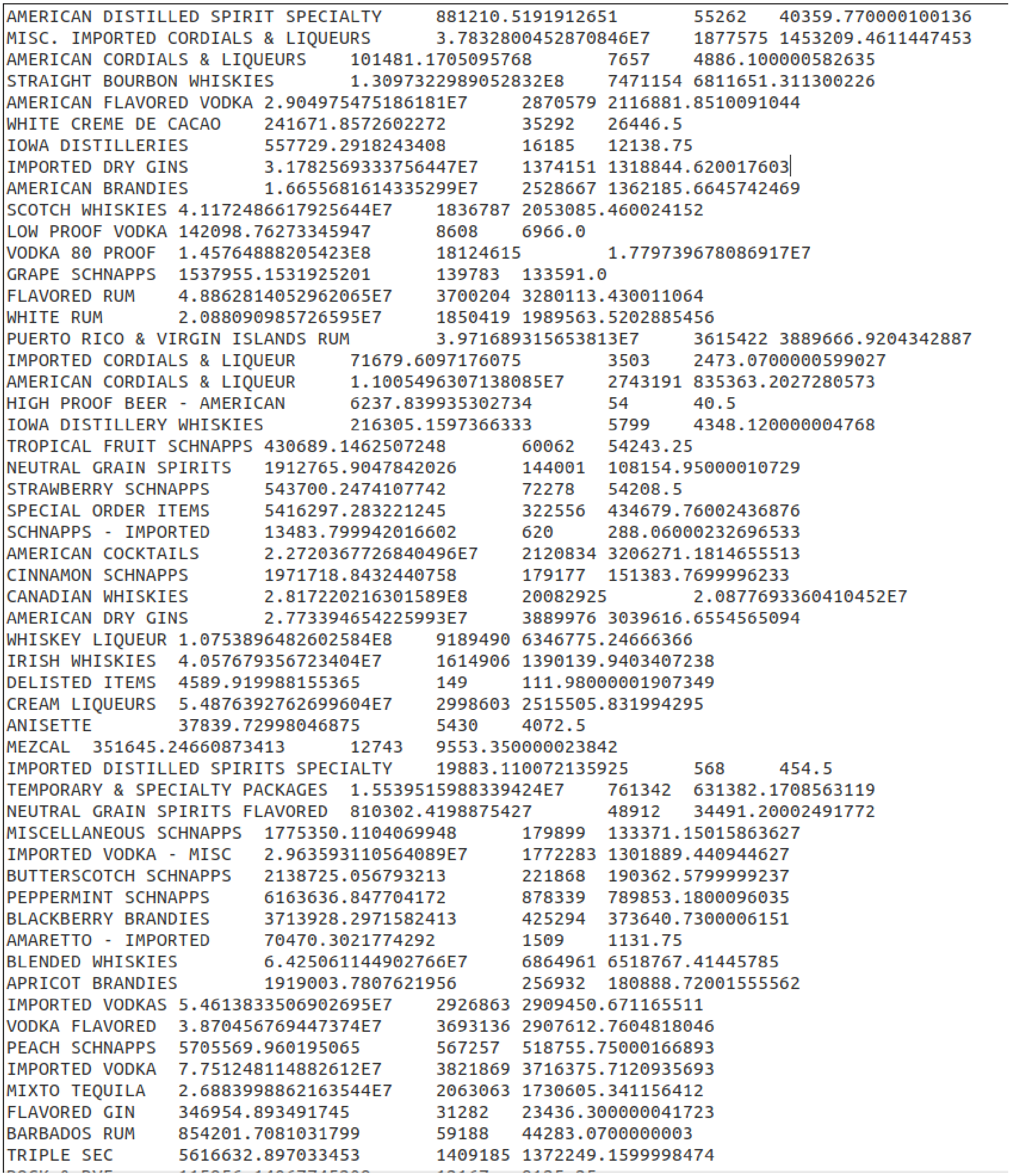
MapReduce Runtime:



```
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=4192925770
        File Output Format Counters
                Bytes Written=0
Time taken for MapReduce Job =1816.814599371 seconds
```

Theoretically and practically Pig is supposed to top MapReduce in terms of processing time and it is quite evident Pig is faster than MapReduce and twice as simple. MapReduce consume approx. 1800 seconds ~ 30 mins as oppose to Pig's 9 minutes

Output:

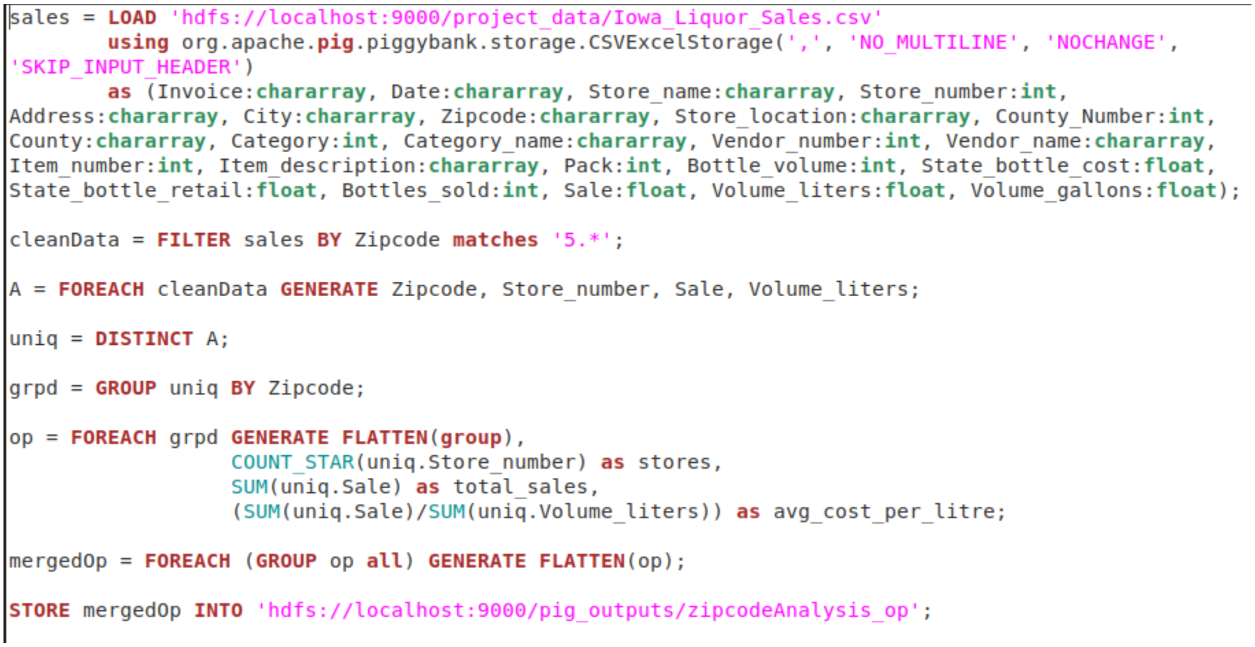


```
AMERICAN DISTILLED SPIRIT SPECIALTY     881210.5191912651       55262   40359.770000100136
MISC. IMPORTED CORDIALS & LIQUEURS      3.7832800452870846E7    1877575 1453209.4611447453
AMERICAN CORDIALS & LIQUEURS    101481.1705095768       7657    4886.100000582635
STRAIGHT BOURBON WHISKIES       1.3097322989052832E8    7471154 6811651.311300226
AMERICAN FLAVORED VODKA 2.904975475186181E7     2870579 2116881.8510091044
WHITE CREME DE CACAO    241671.8572602272       35292   26446.5
IOWA DISTILLERIES       557729.2918243408       16185   12138.75
IMPORTED DRY GINS       3.1782569333756447E7    1374151 1318844.620017603
AMERICAN BRANDIES       1.6655681614335299E7    2528667 1362185.6645742469
SCOTCH WHISKIES 4.1172486617925644E7    1836787 2053085.460024152
LOW PROOF VODKA 142098.76273345947      8608    6966.0
VODKA 80 PROOF  1.45764888205423E8      18124615        1.779739678086917E7
GRAPE SCHNAPPS  1537955.1531925201      139783  133591.0
FLAVORED RUM    4.8862814052962065E7    3700204 3280113.430011064
WHITE RUM       2.0880909857265595E7    1850419 1989563.5202885456
PUERTO RICO & VIRGIN ISLANDS RUM        3.971689315653813E7     3615422 3889666.9204342887
IMPORTED CORDIALS & LIQUEUR     71679.6097176075        3503    2473.0700000599027
AMERICAN CORDIALS & LIQUEUR     1.1005496307138085E7    2743191 835363.2027280573
HIGH PROOF BEER - AMERICAN      6237.839935302734       54      40.5
IOWA DISTILLERY WHISKIES        216305.1597366333       5799    4348.120000004768
TROPICAL FRUIT SCHNAPPS 430689.1462507248       60062   54243.25
NEUTRAL GRAIN SPIRITS   1912765.9047842026      144001  108154.95000010729
STRAWBERRY SCHNAPPS     543700.2474107742       72278   54208.5
SPECIAL ORDER ITEMS     5416297.283221245       322556  434679.76002436876
SCHNAPPS - IMPORTED     13483.799942016602      620     288.06000232696533
AMERICAN COCKTAILS      2.2720367726840496E7    2120834 3206271.1814655513
CINNAMON SCHNAPPS       1971718.8432440758      179177  151383.7699996233
CANADIAN WHISKIES       2.817220216301589E8     20082925        2.0877693360410452E7
AMERICAN DRY GINS       2.773394654225993E7     3889976 3039616.6554565094
WHISKEY LIQUEUR 1.0753896482602584E8    9189490 6346775.24666366
IRISH WHISKIES  4.057679356723404E7     1614906 1390139.9403407238
DELISTED ITEMS  4589.919988155365       149     111.98000001907349
CREAM LIQUEURS  5.4876392762699604E7    2998603 2515505.831994295
ANISETTE        37839.72998046875       5430    4072.5
MEZCAL  351645.24660873413      12743   9553.350000023842
IMPORTED DISTILLED SPIRITS SPECIALTY    19883.110072135925      568     454.5
TEMPORARY & SPECIALTY PACKAGES  1.5539515988339424E7    761342  631382.1708563119
NEUTRAL GRAIN SPIRITS FLAVORED  810302.4198875427       48912   34491.20002491772
MISCELLANEOUS SCHNAPPS  1775350.1104069948      179899  133371.15015863627
IMPORTED VODKA - MISC   2.963593110564089E7     1772283 1301889.440944627
BUTTERSCOTCH SCHNAPPS   2138725.056793213       221868  190362.5799999237
PEPPERMINT SCHNAPPS     6163636.847704172       878339  789853.1800096035
BLACKBERRY BRANDIES     3713928.2971582413      425294  373640.7300006151
AMARETTO - IMPORTED     70470.3021774292        1509    1131.75
BLENDED WHISKIES        6.425061144902766E7     6864961 6518767.41445785
APRICOT BRANDIES        1919003.7807621956      256932  180888.72001555562
IMPORTED VODKAS 5.4613833506902695E7    2926863 2909450.671165511
VODKA FLAVORED  3.870456769447374E7     3693136 2907612.7604818046
PEACH SCHNAPPS  5705569.960195065       567257  518755.75000166893
IMPORTED VODKA  7.751248114882612E7     3821869 3716375.7120935693
MIXTO TEQUILA   2.6883998862163544E7    2063063 1730605.341156412
FLAVORED GIN    346954.893491745        31282   23436.300000041723
BARBADOS RUM    854201.7081031799       59188   44283.0700000003
TRIPLE SEC      5616632.897033453       1409185 1372249.1599998474
```

4. Exploring expansion opportunities using Zip code analysis

The motivation behind this analysis is to provide a comprehensive overview of feasibility of an expansion. If someone is interested in opening a liquor store, one would naturally want to know if he/she is going to sustain and make profit or is there plenty of competition around.

Pig Script:



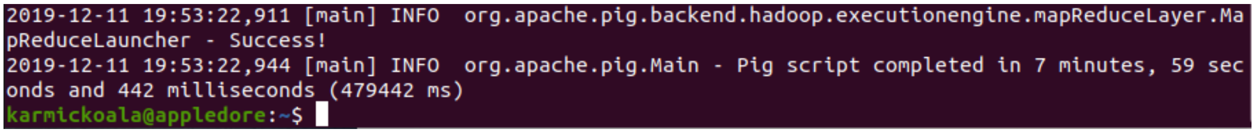
```
sales = LOAD 'hdfs://localhost:9000/project_data/Iowa_Liquor_Sales.csv'
        using org.apache.pig.piggybank.storage.CSVExcelStorage(',', 'NO_MULTILINE', 'NOCHANGE',
'SKIP_INPUT_HEADER')
        as (Invoice:chararray, Date:chararray, Store_name:chararray, Store_number:int,
Address:chararray, City:chararray, Zipcode:chararray, Store_location:chararray, County_Number:int,
County:chararray, Category:int, Category_name:chararray, Vendor_number:int, Vendor_name:chararray,
Item_number:int, Item_description:chararray, Pack:int, Bottle_volume:int, State_bottle_cost:float,
State_bottle_retail:float, Bottles_sold:int, Sale:float, Volume_liters:float, Volume_gallons:float);

cleanData = FILTER sales BY Zipcode matches '5.*';

A = FOREACH cleanData GENERATE Zipcode, Store_number, Sale, Volume_liters;

uniq = DISTINCT A;

grpd = GROUP uniq BY Zipcode;

op = FOREACH grpd GENERATE FLATTEN(group),
                  COUNT_STAR(uniq.Store_number) as stores,
                  SUM(uniq.Sale) as total_sales,
                  (SUM(uniq.Sale)/SUM(uniq.Volume_liters)) as avg_cost_per_litre;

mergedOp = FOREACH (GROUP op all) GENERATE FLATTEN(op);

STORE mergedOp INTO 'hdfs://localhost:9000/pig_outputs/zipcodeAnalysis_op';
```

Here, I encountered a couple of zip codes namely, 80904: Colorado & 87325: New Mexico. In order to skip such dirty data, we perform filtering before we group data by Zip code.

Additionally, we want to get a count of stores within a zip code. As our data is on a transactional level, simply taking a count can result in incorrect values. We want distinct number of stores within a zip code where transactions have taken place. For this, we get a distinct of zip codes.

Since, some rows have nulls in place of store number, we take a COUNT_STAR that considers nulls but does not include it in the count.

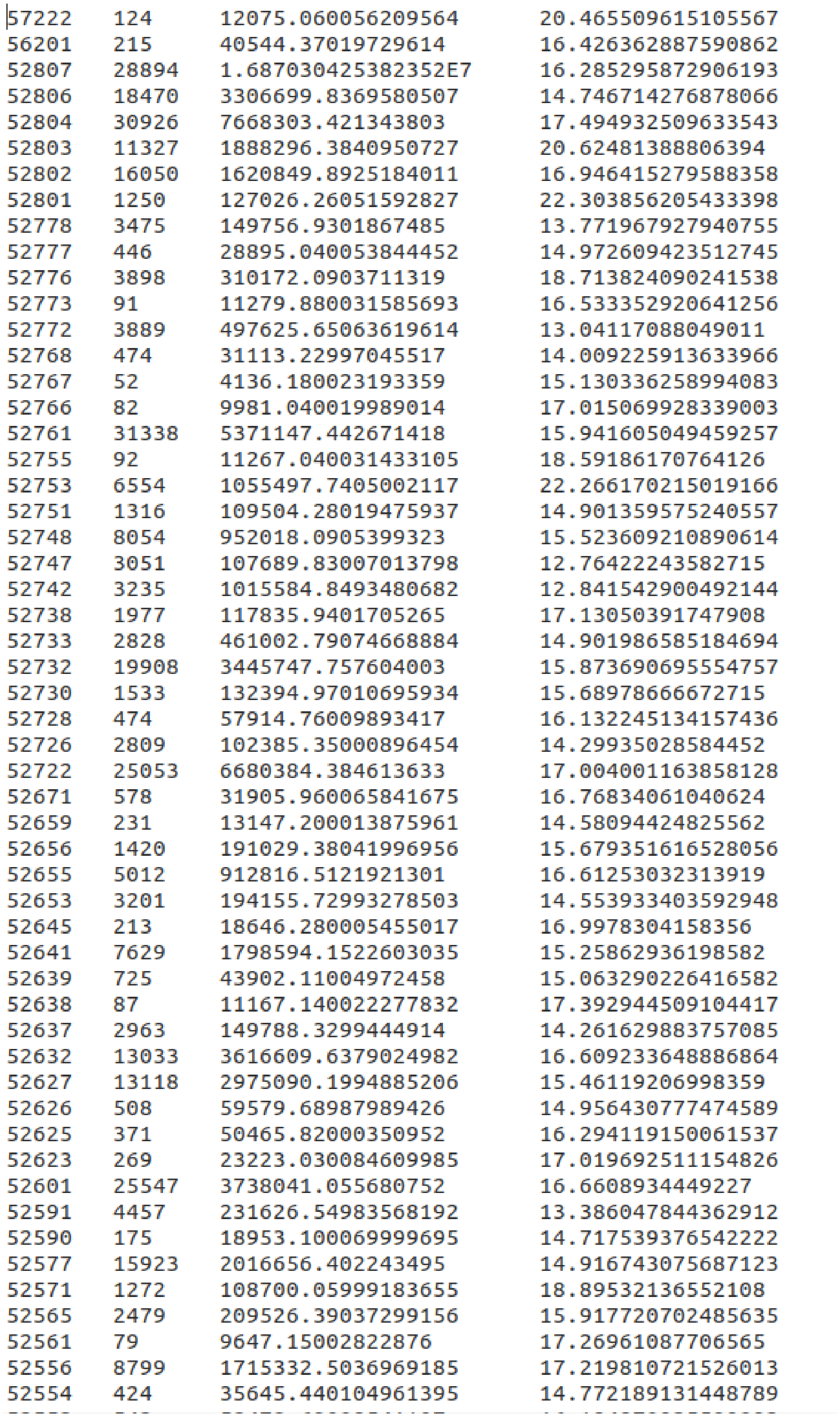
PIG Runtime:



```
2019-12-11 19:53:22,911 [main] INFO  org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.Ma
pReduceLauncher - Success!
2019-12-11 19:53:22,944 [main] INFO  org.apache.pig.Main - Pig script completed in 7 minutes, 59 sec
onds and 442 milliseconds (479442 ms)
karmickoala@appledore:~$
```

Once again, Pig has topped its previous best with 7 minutes this time.

Output:



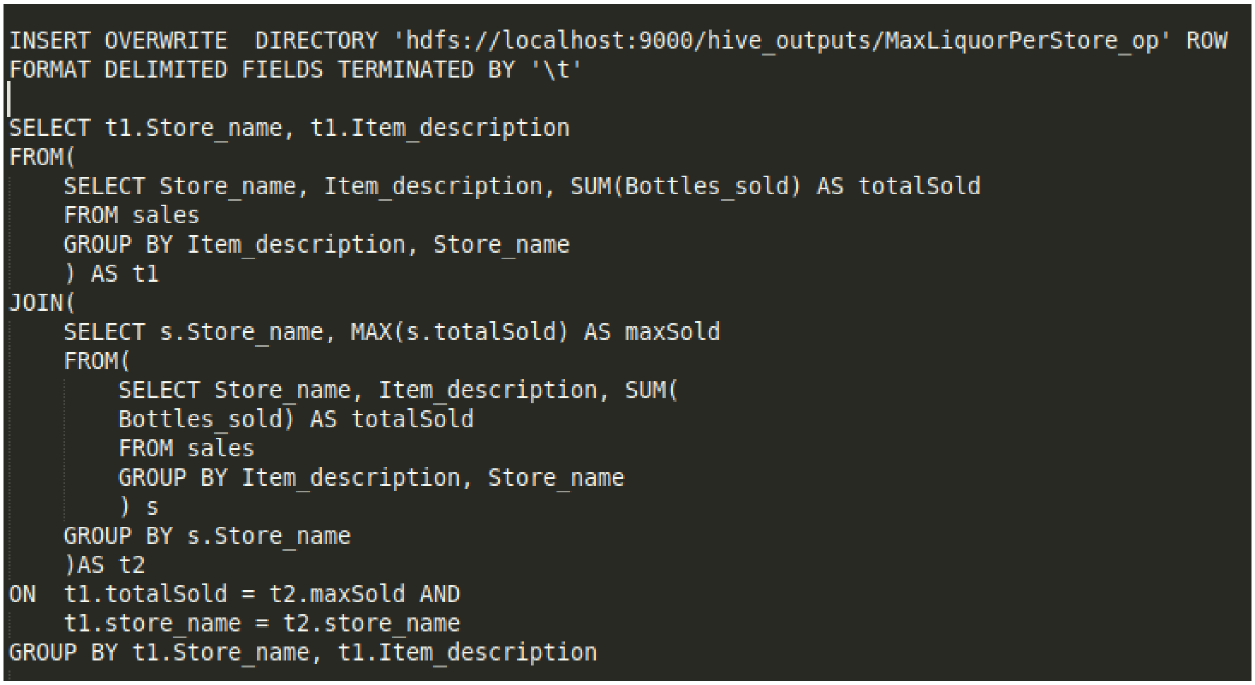
```
57222   124     12075.060056209564      20.465509615105567
56201   215     40544.37019729614       16.426362887590862
52807   28894   1.687030425382352E7     16.285295872906193
52806   18470   3306699.8369580507      14.746714276878066
52804   30926   7668303.421343803       17.494932509633543
52803   11327   1888296.3840950727      20.62481388806394
52802   16050   1620849.8925184011      16.946415279588358
52801   1250    127026.26051592827      22.303856205433398
52778   3475    149756.9301867485       13.771967927940755
52777   446     28895.040053844452      14.972609423512745
52776   3898    310172.0903711319       18.713824090241538
52773   91      11279.880031585693      16.533352920641256
52772   3889    497625.65063619614      13.04117088049011
52768   474     31113.22997045517       14.009225913633966
52767   52      4136.180023193359       15.130336258994083
52766   82      9981.040019989014       17.015069928339003
52761   31338   5371147.442671418       15.941605049459257
52755   92      11267.040031433105      18.59186170764126
52753   6554    1055497.7405002117      22.266170215019166
52751   1316    109504.28019475937      14.901359575240557
52748   8054    952018.0905399323       15.523609210890614
52747   3051    107689.83007013798      12.76422243582715
52742   3235    1015584.8493480682      12.841542900492144
52738   1977    117835.9401705265       17.13050391747908
52733   2828    461002.79074668884      14.901986585184694
52732   19908   3445747.757604003       15.873690695554757
52730   1533    132394.97010695934      15.68978666672715
52728   474     57914.76009893417       16.132245134157436
52726   2809    102385.35000896454      14.29935028584452
52722   25053   6680384.384613633       17.004001163858128
52671   578     31905.960065841675      16.76834061040624
52659   231     13147.200013875961      14.58094424825562
52656   1420    191029.38041996956      15.679351616528056
52655   5012    912816.5121921301       16.61253032313919
52653   3201    194155.72993278503      14.553933403592948
52645   213     18646.280005455017      16.9978304158356
52641   7629    1798594.1522603035      15.25862936198582
52639   725     43902.11004972458       15.063290226416582
52638   87      11167.140022277832      17.392944509104417
52637   2963    149788.3299444914       14.261629883757085
52632   13033   3616609.6379024982      16.609233648886864
52627   13118   2975090.1994885206      15.46119206998359
52626   508     59579.68987989426       14.956430777474589
52625   371     50465.82000350952       16.294119150061537
52623   269     23223.030084609985      17.019692511154826
52601   25547   3738041.055680752       16.6608934449227
52591   4457    231626.54983568192      13.386047844362912
52590   175     18953.100069999695      14.717539376542222
52577   15923   2016656.402243495       14.916743075687123
52571   1272    108700.05999183655      18.89532136552108
52565   2479    209526.39037299156      15.917720702485635
52561   79      9647.15002822876        17.26961087706565
52556   8799    1715332.5036969185      17.219810721526013
52554   424     35645.440104961395      14.772189131448789
```

5. Researching top alcohol brand sold by each store in the state

The intent behind this use case was to understand the popularity of liquor by stores across the states. Any store near a college, university or any institution that has young students around would sell a lot of beers or alcopops. Similarly, stores right in middle of a community will cater to a larger older population and may sell hard liquors as opposed to beers.

This supplements well to the 2nd use case where we found out top 5 brands per year based on sales. It might also help in stocking inventory.

Hive Script:

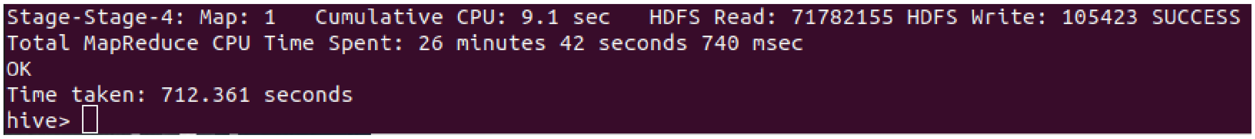


```
INSERT OVERWRITE  DIRECTORY 'hdfs://localhost:9000/hive_outputs/MaxLiquorPerStore_op' ROW
FORMAT DELIMITED FIELDS TERMINATED BY '\t'

SELECT t1.Store_name, t1.Item_description
FROM(
    SELECT Store_name, Item_description, SUM(Bottles_sold) AS totalSold
    FROM sales
    GROUP BY Item_description, Store_name
    ) AS t1
JOIN(
    SELECT s.Store_name, MAX(s.totalSold) AS maxSold
    FROM(
        SELECT Store_name, Item_description, SUM(
        Bottles_sold) AS totalSold
        FROM sales
        GROUP BY Item_description, Store_name
        ) s
    GROUP BY s.Store_name
    )AS t2
ON  t1.totalSold = t2.maxSold AND
    t1.store_name = t2.store_name
GROUP BY t1.Store_name, t1.Item_description
```

The SQL here is self-explanatory.

1. First, we get the store name, item description and total sum of bottles sold which is our metric
2. Then, we get the value maximum bottles have been sold in a store
3. Join both queries and where ever the maximum count matches the sum of bottles sold for an alcohol, that's our most popular liquor for that store

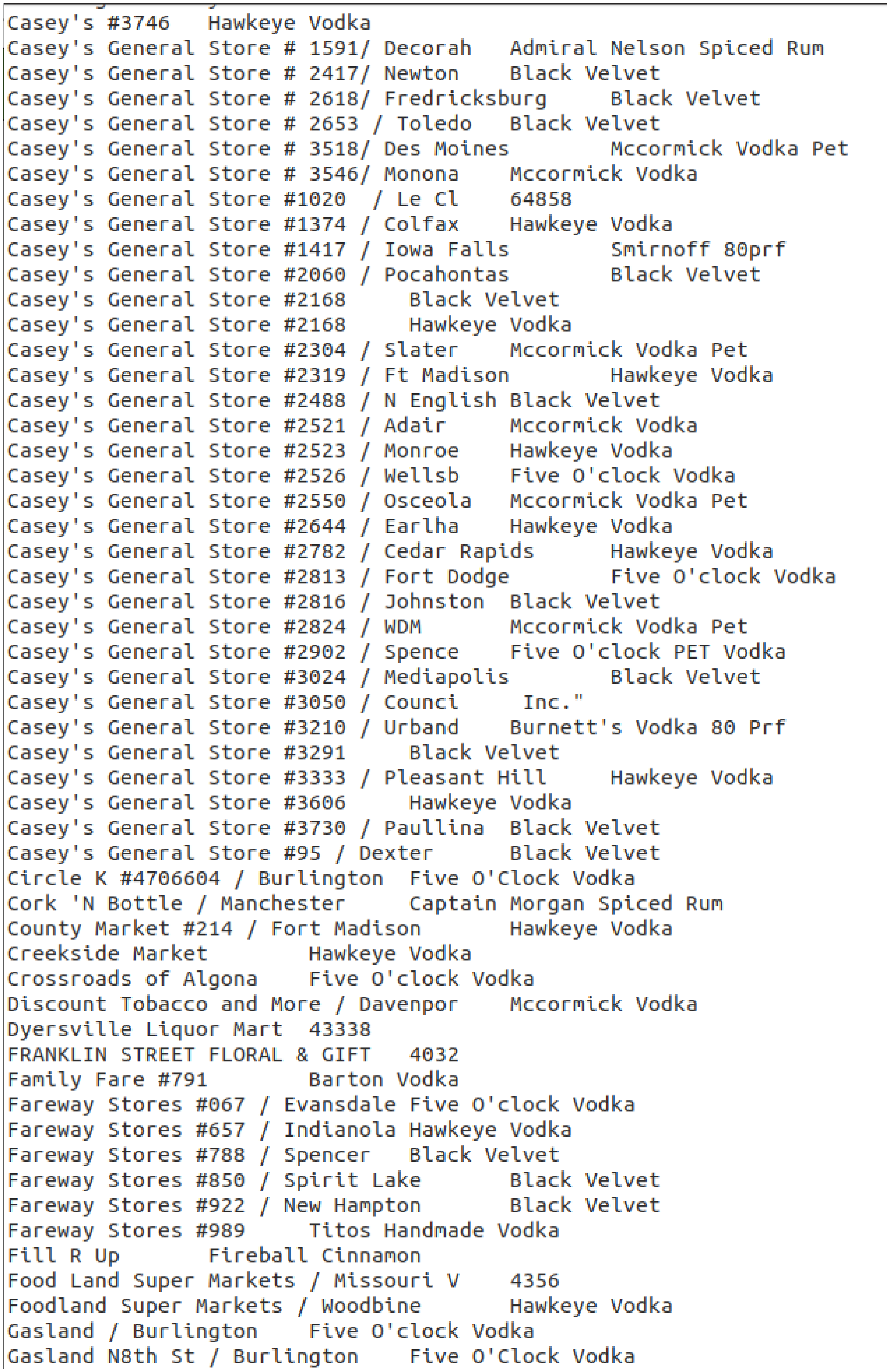
Hive Run time:



```
Stage-Stage-4: Map: 1   Cumulative CPU: 9.1 sec   HDFS Read: 71782155 HDFS Write: 105423 SUCCESS
Total MapReduce CPU Time Spent: 26 minutes 42 seconds 740 msec
OK
Time taken: 712.361 seconds
hive> 
```

Hive seems to be pretty fast as well. 712 seconds ~ 12 minutes to crunch almost 17 million records. Impressive!

Output:



```
Casey's #3746   Hawkeye Vodka
Casey's General Store # 1591/ Decorah   Admiral Nelson Spiced Rum
Casey's General Store # 2417/ Newton    Black Velvet
Casey's General Store # 2618/ Fredricksburg     Black Velvet
Casey's General Store # 2653 / Toledo   Black Velvet
Casey's General Store # 3518/ Des Moines        Mccormick Vodka Pet
Casey's General Store # 3546/ Monona    Mccormick Vodka
Casey's General Store #1020  / Le Cl    64858
Casey's General Store #1374 / Colfax    Hawkeye Vodka
Casey's General Store #1417 / Iowa Falls        Smirnoff 80prf
Casey's General Store #2060 / Pocahontas        Black Velvet
Casey's General Store #2168     Black Velvet
Casey's General Store #2168     Hawkeye Vodka
Casey's General Store #2304 / Slater    Mccormick Vodka Pet
Casey's General Store #2319 / Ft Madison        Hawkeye Vodka
Casey's General Store #2488 / N English Black Velvet
Casey's General Store #2521 / Adair     Mccormick Vodka
Casey's General Store #2523 / Monroe    Hawkeye Vodka
Casey's General Store #2526 / Wellsb    Five O'clock Vodka
Casey's General Store #2550 / Osceola   Mccormick Vodka Pet
Casey's General Store #2644 / Earlha    Hawkeye Vodka
Casey's General Store #2782 / Cedar Rapids      Hawkeye Vodka
Casey's General Store #2813 / Fort Dodge        Five O'clock Vodka
Casey's General Store #2816 / Johnston  Black Velvet
Casey's General Store #2824 / WDM       Mccormick Vodka Pet
Casey's General Store #2902 / Spence    Five O'clock PET Vodka
Casey's General Store #3024 / Mediapolis        Black Velvet
Casey's General Store #3050 / Counci     Inc."
Casey's General Store #3210 / Urband    Burnett's Vodka 80 Prf
Casey's General Store #3291     Black Velvet
Casey's General Store #3333 / Pleasant Hill     Hawkeye Vodka
Casey's General Store #3606     Hawkeye Vodka
Casey's General Store #3730 / Paullina  Black Velvet
Casey's General Store #95 / Dexter      Black Velvet
Circle K #4706604 / Burlington  Five O'Clock Vodka
Cork 'N Bottle / Manchester     Captain Morgan Spiced Rum
County Market #214 / Fort Madison       Hawkeye Vodka
Creekside Market        Hawkeye Vodka
Crossroads of Algona    Five O'clock Vodka
Discount Tobacco and More / Davenpor    Mccormick Vodka
Dyersville Liquor Mart    43338
FRANKLIN STREET FLORAL & GIFT   4032
Family Fare #791        Barton Vodka
Fareway Stores #067 / Evansdale Five O'clock Vodka
Fareway Stores #657 / Indianola Hawkeye Vodka
Fareway Stores #788 / Spencer   Black Velvet
Fareway Stores #850 / Spirit Lake       Black Velvet
Fareway Stores #922 / New Hampton       Black Velvet
Fareway Stores #989     Titos Handmade Vodka
Fill R Up       Fireball Cinnamon
Food Land Super Markets / Missouri V    4356
Foodland Super Markets / Woodbine       Hawkeye Vodka
Gasland / Burlington    Five O'clock Vodka
Gasland N8th St / Burlington    Five O'Clock Vodka
```