IDENTIFICATION OF TOP -10 PLATFORM AND GENRE FOR GAMES USING HADOOP

ABSTRACT

This report presents a project that utilizes Hadoop MapReduce chaining to identify the top 10 video game platform and genre combinations based on sales data. By leveraging Hadoop's MapReduce framework, the project aims to showcase its potential in handling large-scale datasets and emphasize the significance of data analysis in the gaming industry. The project involves mapping video game sales records to identify platform and genre attributes, followed by aggregating the sales data for each combination. A second MapReduce chain is employed to sort the aggregated data and determine the top 10 combinations, providing valuable insights for game developers, marketers, and investors. The project highlights Hadoop's scalability, flexibility, and its role in facilitating data-driven decision-making in the gaming industry.

Tanvi Mathur
Data Science & Analytics Student

Introduction

Map Reduce is a Hadoop data analysis technique that allows data to expand and be managed across several computer nodes. It filters and turns input into a value that the reducer may aggregate overusing basic units called as mappers and reducers. Lists and key/value pairs are the basics of map reduction, and they serve as constraints for the map reduce method. The programme takes a list of key-value pairs as input (). Following the scanning of the input file, mapper's map function divides each line into a key value pair. The output is added to a long list of pairs, which are then joined to form a new pair. With this design, the reducer evaluates each component individually. The MapReduce framework's primary features are its simplicity, scalability, and extensibility, which allow it to process huge amounts of information (Lam, 2011).

(Input) -> <k1,v1> -> map -> < k2,v2> -> reduce -> <k3,v3> -> (Output)

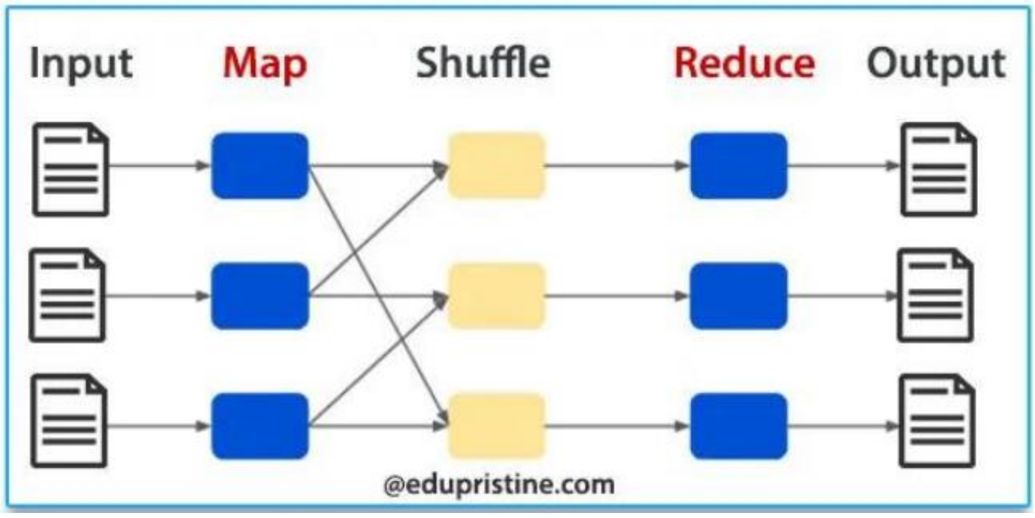


Figure2: MapReduce Steps

Problem Statement and Data:

Firstly, the problem statement for this report is "***What are the top 10 genre console combination which are popular among the buyers based on total sales?***", the objective of this problem statement is to analyse the global sales for each genre of video games across different gaming consoles. The analysis will help us understand which genre-console combinations are most popular among buyers. This information can help game developers to focus on popular combinations, improve game quality and drive sales revenue. The dataset used for the analysis was obtained from the Kaggle website which comprises of 13764 rows and 14 columns. The collection of data is widely ranged from the year 2000 to 2020. In the below table, each parameter is extensively described in the data lexicon.

Parameters	Synopsis	Data Type
Title	Name of the game	Categorical
Console	Platform of the game	Categorical
Genre	Genre of the game	Categorical
Publisher	Publisher of the game	Categorical
Vg_score	The vgchartz critical score (out of 10)	Float
Critic_score	The metacritic critical score (out of 10)	Float
User_score	The user critical score (out of 10)	Float
Total_shipped	The total number shipped in millions	Float
Asian_sales	Asian region sales in millions	Float
North_American_sales	North American region sales in millions	Float
Japan_sales	Japan region sales in millions	Float
European_sales	European region sales in millions	Float
Global_sales	Worldwide sales in millions	Float
Release_year	Year game was released	Numeric

Table1: Description of video game sales data

Design & Implementation:

For this project, we will create a script in Eclipse using the programming language 'Java,' then export it to see what output is created when such code is executed on the dataset. These outputs will then serve as the foundation for the report's findings section. This problem statement can be answered using the **MapReduce chaining**, to obtain the top 10 genre console combinations with highest sales which will help developers identify the popular gaming combination which they should consider while creating new games. The flow of the solution is such that we are going to run two MapReduce jobs in sequence, thus the output of first job will be an input for the second job.

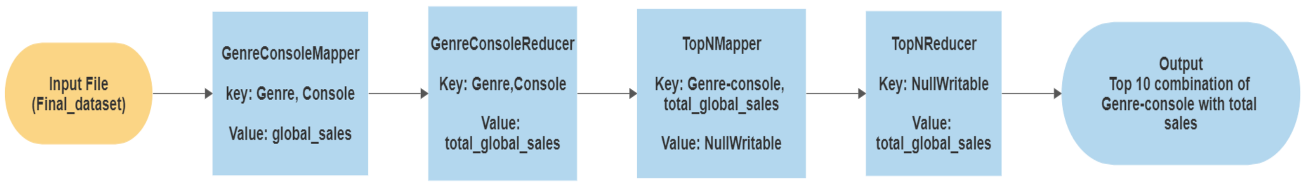

Input File (Final_dataset)
GenreConsoleMapper
key: Genre, Console
Value: global_sales
GenreConsoleReducer
Key: Genre,Console
Value: total_global_sales
TopNMapper
Key: Genre-console, total_global_sales
Value: NullWritable
TopNReducer
Key: NullWritable
Value: total_global_sales
Output
Top 10 combination of Genre-console with total sales

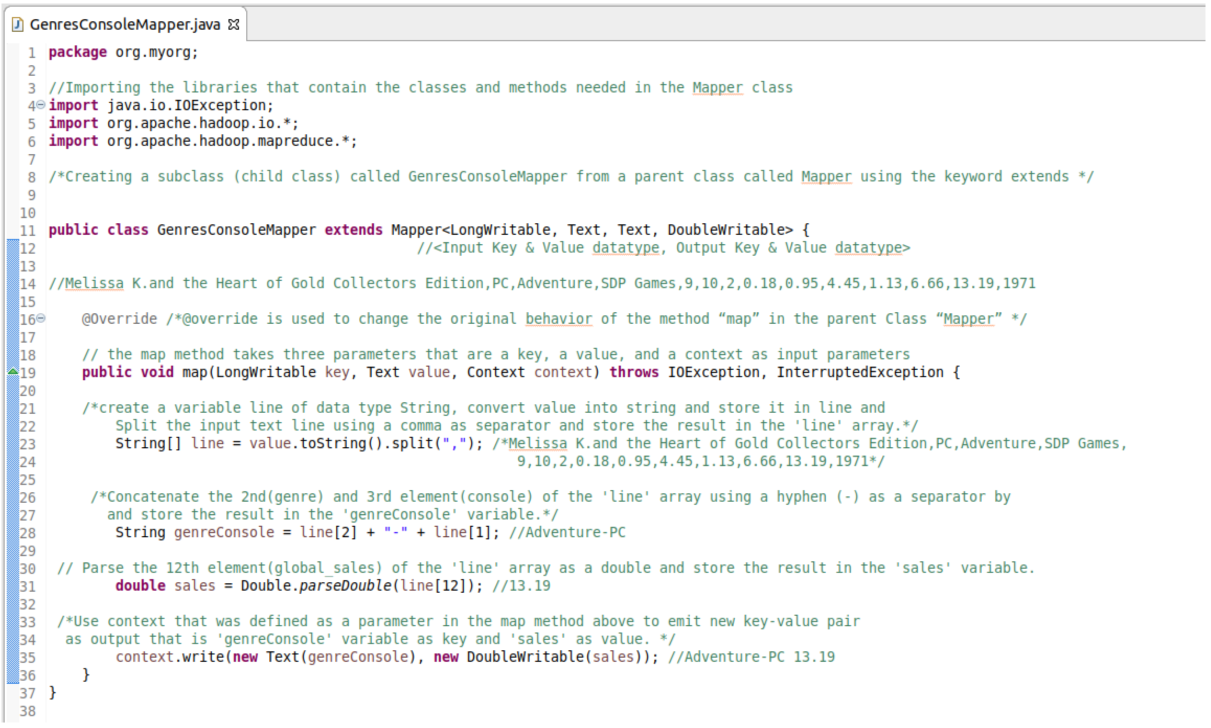
Figure 3: Flow diagram

Let go ahead and have a look at the different files we created in Eclipse to run the MapReduce flow and achieve the desired outcome.

There are a total of 5 java classes in our MapReduce project namely:

1. GenresConsoleMapper – The first Mapper class
2. GenresConsoleReducer – The first Reducer class
3. TopNMapper– The second Mapper class
4. TopNReducer– The second Reducer class
5. TopGenresConsoleSales – The main driver class
6. Final_dataset – The input text file

Mapper 1: GenreConsoleMapper



```java
package org.myorg;

//Importing the libraries that contain the classes and methods needed in the Mapper class
import java.io.IOException;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

/*Creating a subclass (child class) called GenresConsoleMapper from a parent class called Mapper using the keyword extends */


public class GenresConsoleMapper extends Mapper<LongWritable, Text, Text, DoubleWritable> {
                                            //<Input Key & Value datatype, Output Key & Value datatype>

//Melissa K.and the Heart of Gold Collectors Edition,PC,Adventure,SDP Games,9,10,2,0.18,0.95,4.45,1.13,6.66,13.19,1971

    @Override /*@override is used to change the original behavior of the method “map” in the parent Class “Mapper” */

    // the map method takes three parameters that are a key, a value, and a context as input parameters
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

    /*create a variable line of data type String, convert value into string and store it in line and
        Split the input text line using a comma as separator and store the result in the 'line' array.*/
        String[] line = value.toString().split(","); /*Melissa K.and the Heart of Gold Collectors Edition,PC,Adventure,SDP Games,
                                                     9,10,2,0.18,0.95,4.45,1.13,6.66,13.19,1971*/

     /*Concatenate the 2nd(genre) and 3rd element(console) of the 'line' array using a hyphen (-) as a separator by
       and store the result in the 'genreConsole' variable.*/
        String genreConsole = line[2] + "-" + line[1]; //Adventure-PC

 // Parse the 12th element(global_sales) of the 'line' array as a double and store the result in the 'sales' variable.
        double sales = Double.parseDouble(line[12]); //13.19

 /*Use context that was defined as a parameter in the map method above to emit new key-value pair
  as output that is 'genreConsole' variable as key and 'sales' as value. */
        context.write(new Text(genreConsole), new DoubleWritable(sales)); //Adventure-PC 13.19
    }
}
```

Figure 5: GenreConsoleMapper

The First mapper deal with the input from the text file it will read the input as:

```
// Melissa K.and the Heart of Gold Collectors Edition,PC,Adventure,SDP Games,9,10,2,0.18,0.95,4.45,1.13,6.66,13.19,1971
```

In the map method, we must take a close look at our key and value. Since the analysis, we must calculate for the genre and console combination, Genre and Console will be the key. Since we are interested in finding total global sales for combination the global_sales will be value.

Line 22 splits the data and stores it in a array. Next the Line 26 here from the above data genre and console are picked and concatenated with a hyphen as shown below.

```
//Adventure-PC
```

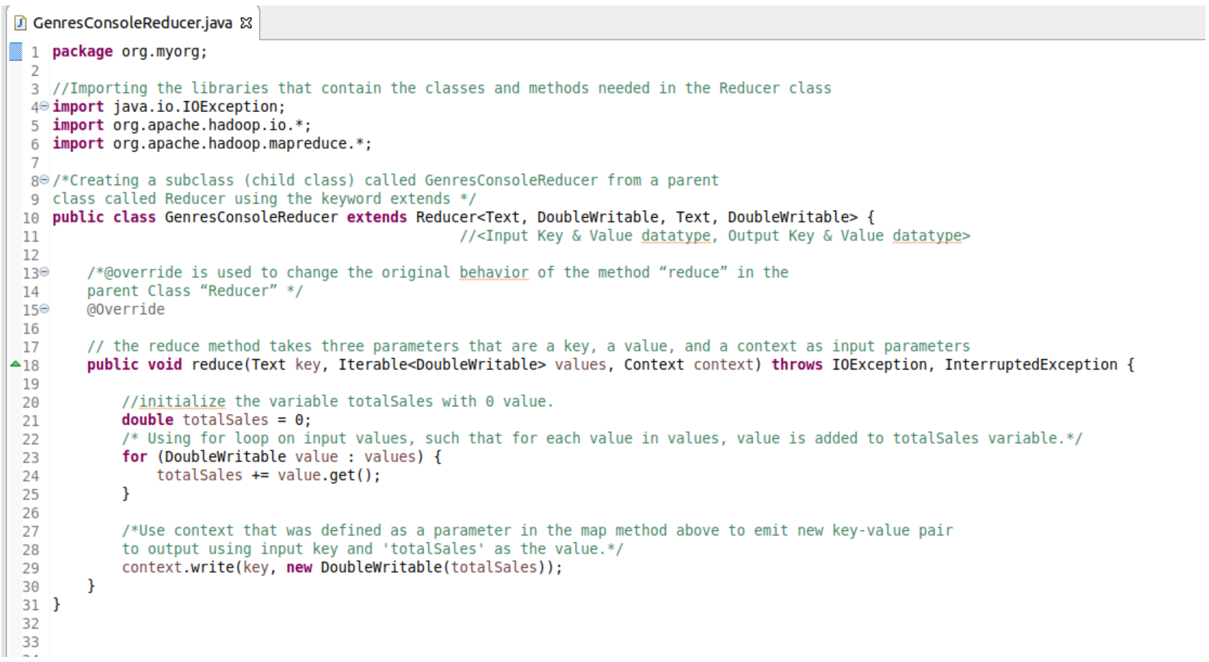
While line 29 fetch the data which is present at 12th index in the input file that is shown below.

```
//13.19
```

Finally in line 33 the data is passed as key value pair to the reducer where here the key is Adventure-PC and 13.19 is value as shown below.

```
//Adventure-PC  13.19
```

Reducer 1: GenreConsoleReducer

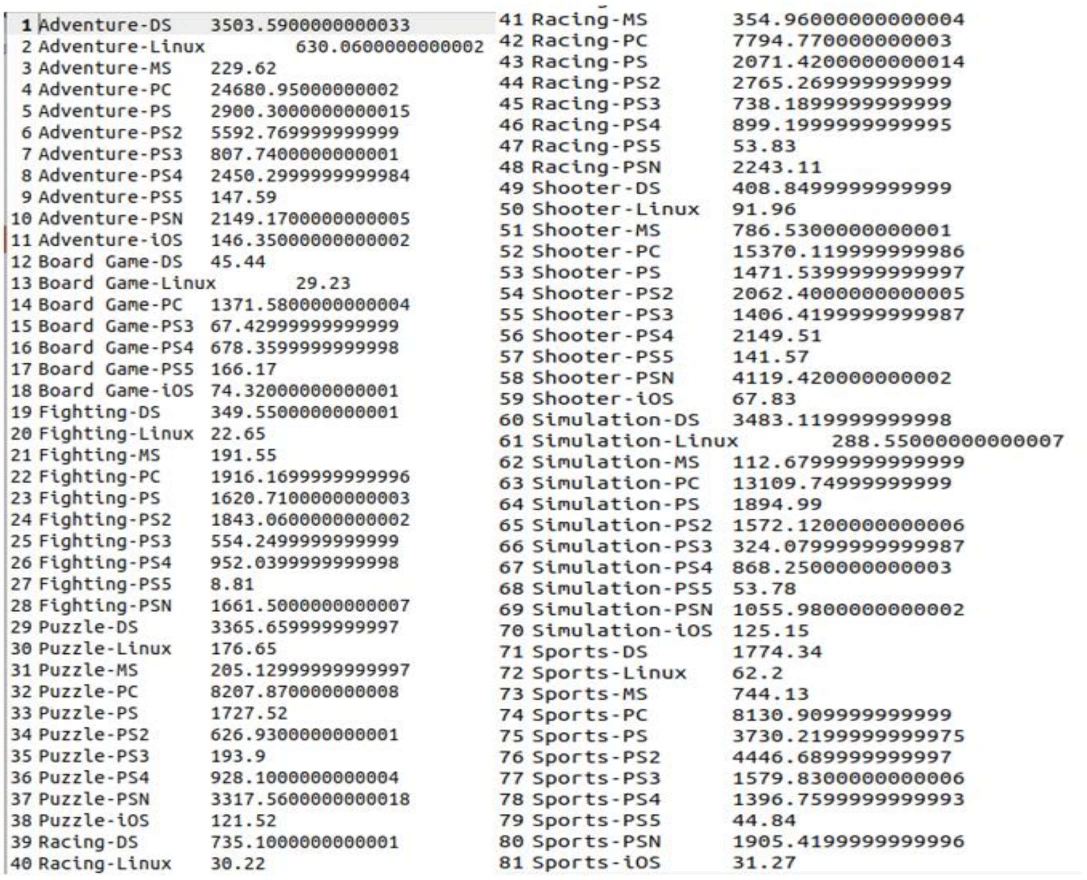


GenresConsoleReducer.java

```
1  package org.myorg;
2
3  //Importing the libraries that contain the classes and methods needed in the Reducer class
4  import java.io.IOException;
5  import org.apache.hadoop.io.*;
6  import org.apache.hadoop.mapreduce.*;
7
8  /*Creating a subclass (child class) called GenresConsoleReducer from a parent
9  class called Reducer using the keyword extends */
10 public class GenresConsoleReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
11                                                //<Input Key & Value datatype, Output Key & Value datatype>
12
13     /*@override is used to change the original behavior of the method “reduce” in the
14     parent Class “Reducer” */
15     @Override
16
17     // the reduce method takes three parameters that are a key, a value, and a context as input parameters
18     public void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws IOException, InterruptedException {
19
20         //initialize the variable totalSales with 0 value.
21         double totalSales = 0;
22         /* Using for loop on input values, such that for each value in values, value is added to totalSales variable.*/
23         for (DoubleWritable value : values) {
24             totalSales += value.get();
25         }
26
27         /*Use context that was defined as a parameter in the map method above to emit new key-value pair
28         to output using input key and 'totalSales' as the value.*/
29         context.write(key, new DoubleWritable(totalSales));
30     }
31 }
32
33
```

Figure 6: GenreConsoleReducer

As we go to the reducer in Line 10 key is passed as [{Adventure-PC,13.19}, {Adventure-PC,6.89},}, {Adventure-PC,11.36},},{Puzzel-DS,9.56}]. The line 22 is a for loop which based on the genre console combination value help generate the total sales. From line 28 we will get total sales for different key, for instance [{Adventure-PC 31.44}]. So overall, the first MapReduce generates the total sales for each combination of genre and console. The output file for first MapReduce is shown below.

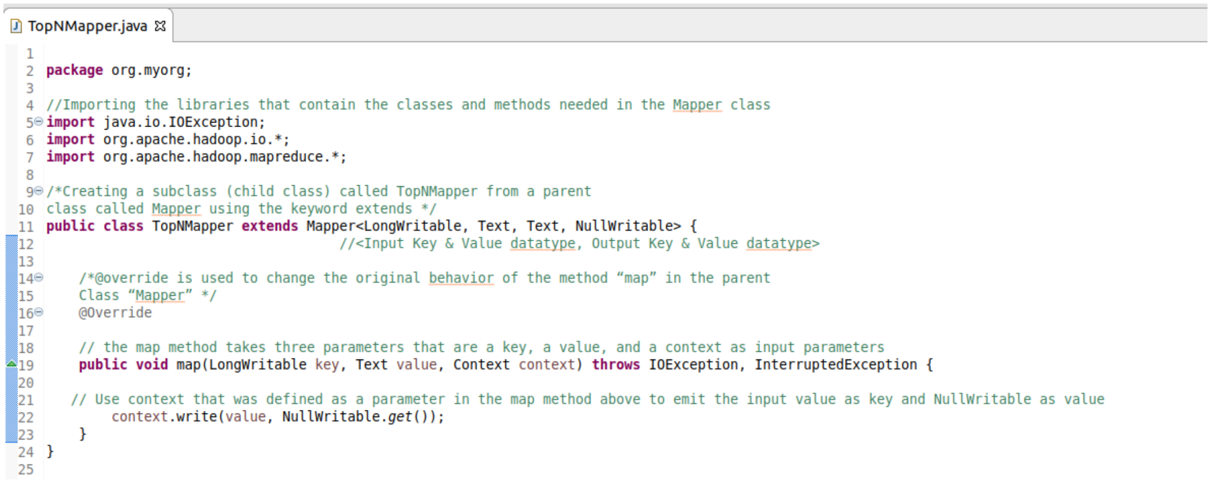


```
1 Adventure-DS	3503.5900000000033
2 Adventure-Linux	630.0600000000002
3 Adventure-MS	229.62
4 Adventure-PC	24680.95000000002
5 Adventure-PS	2900.3000000000015
6 Adventure-PS2	5592.769999999999
7 Adventure-PS3	807.7400000000001
8 Adventure-PS4	2450.2999999999984
9 Adventure-PS5	147.59
10 Adventure-PSN	2149.1700000000005
11 Adventure-iOS	146.35000000000002
12 Board Game-DS	45.44
13 Board Game-Linux	29.23
14 Board Game-PC	1371.5800000000004
15 Board Game-PS3	67.42999999999999
16 Board Game-PS4	678.3599999999998
17 Board Game-PS5	166.17
18 Board Game-iOS	74.32000000000001
19 Fighting-DS	349.5500000000001
20 Fighting-Linux	22.65
21 Fighting-MS	191.55
22 Fighting-PC	1916.1699999999996
23 Fighting-PS	1620.7100000000003
24 Fighting-PS2	1843.0600000000002
25 Fighting-PS3	554.2499999999999
26 Fighting-PS4	952.0399999999998
27 Fighting-PS5	8.81
28 Fighting-PSN	1661.5000000000007
29 Puzzle-DS	3365.659999999997
30 Puzzle-Linux	176.65
31 Puzzle-MS	205.12999999999997
32 Puzzle-PC	8207.870000000008
33 Puzzle-PS	1727.52
34 Puzzle-PS2	626.9300000000001
35 Puzzle-PS3	193.9
36 Puzzle-PS4	928.1000000000004
37 Puzzle-PSN	3317.5600000000018
38 Puzzle-iOS	121.52
39 Racing-DS	735.1000000000001
40 Racing-Linux	30.22
41 Racing-MS	354.96000000000004
42 Racing-PC	7794.770000000003
43 Racing-PS	2071.4200000000014
44 Racing-PS2	2765.269999999999
45 Racing-PS3	738.1899999999999
46 Racing-PS4	899.1999999999995
47 Racing-PS5	53.83
48 Racing-PSN	2243.11
49 Shooter-DS	408.8499999999999
50 Shooter-Linux	91.96
51 Shooter-MS	786.5300000000001
52 Shooter-PC	15370.119999999986
53 Shooter-PS	1471.5399999999997
54 Shooter-PS2	2062.4000000000005
55 Shooter-PS3	1406.4199999999987
56 Shooter-PS4	2149.51
57 Shooter-PS5	141.57
58 Shooter-PSN	4119.420000000002
59 Shooter-iOS	67.83
60 Simulation-DS	3483.119999999998
61 Simulation-Linux	288.55000000000007
62 Simulation-MS	112.67999999999999
63 Simulation-PC	13109.74999999999
64 Simulation-PS	1894.99
65 Simulation-PS2	1572.1200000000006
66 Simulation-PS3	324.07999999999987
67 Simulation-PS4	868.2500000000003
68 Simulation-PS5	53.78
69 Simulation-PSN	1055.9800000000002
70 Simulation-iOS	125.15
71 Sports-DS	1774.34
72 Sports-Linux	62.2
73 Sports-MS	744.13
74 Sports-PC	8130.909999999999
75 Sports-PS	3730.2199999999975
76 Sports-PS2	4446.689999999997
77 Sports-PS3	1579.8300000000006
78 Sports-PS4	1396.7599999999993
79 Sports-PS5	44.84
80 Sports-PSN	1905.4199999999996
81 Sports-iOS	31.27
```

Figure 7: Output File for GenresConsole MapReduce

The output from first MapReduce is all possible combinations of genre and console with their total sales. So, we can make out that there are a total of 81 combinations, and each have a different sale which basically indicates the popularity of them. Going further in the next MapReduce the outcome is further processed to receive the top 10 combinations based on sales.

Mapper 2: TopNMapper



```java

package org.myorg;

//Importing the libraries that contain the classes and methods needed in the Mapper class
import java.io.IOException;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

/*Creating a subclass (child class) called TopNMapper from a parent
class called Mapper using the keyword extends */
public class TopNMapper extends Mapper<LongWritable, Text, Text, NullWritable> {
                                   //<Input Key & Value datatype, Output Key & Value datatype>

    /*@override is used to change the original behavior of the method “map” in the parent
    Class “Mapper” */
    @Override

    // the map method takes three parameters that are a key, a value, and a context as input parameters
    public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {

   // Use context that was defined as a parameter in the map method above to emit the input value as key and NullWritable as value
        context.write(value, NullWritable.get());
    }
}

```

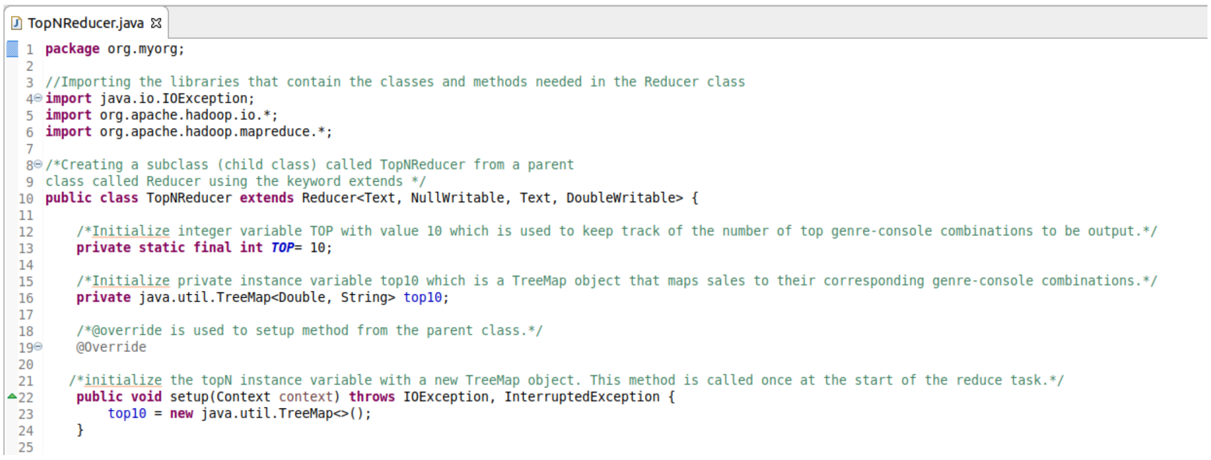
Figure 8: TopNMapper

The input for the second mapper is the output from **GenresConsoleReducer**. It is read as shown below.

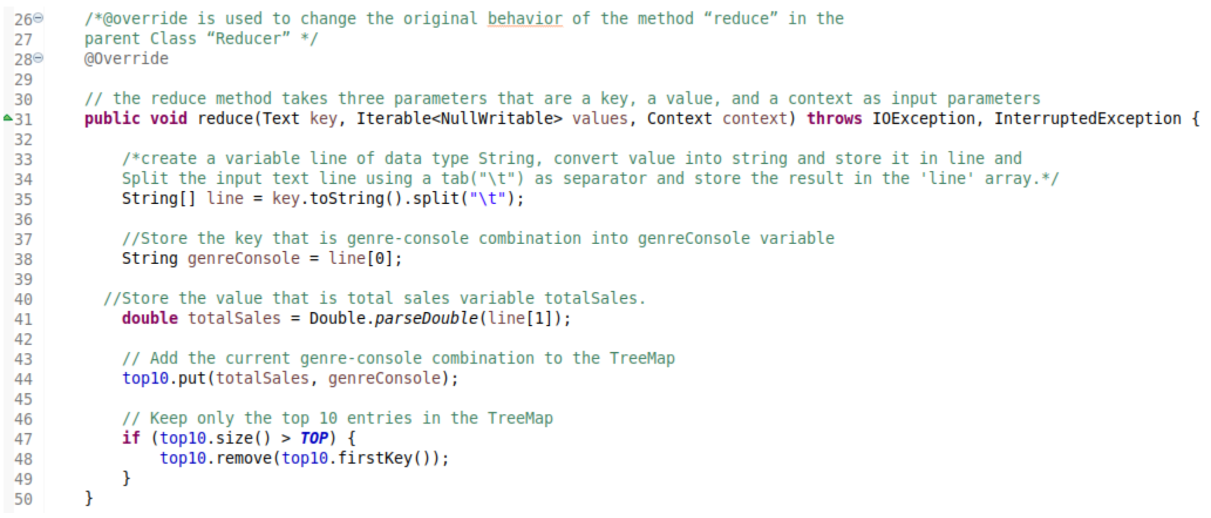
```
/*[{Adventure-DS    3503.5900000000033} {Adventure-Linux    630.0600000000002} {Adventure-MS    229.62} {Adventure-PC    24680.95000000002}]*/
```

LongWritable is a byte offset from the current line in the input file, Text is the actual data on the line, and Context is used to emit intermediate key-value pairs. In line 18 the value input is written as the key output with a NullWritable value. This means that for each input line, the mapper will output a key-value pair where the key is the input line, and the value is NullWritable which is then passed to the next reducer.

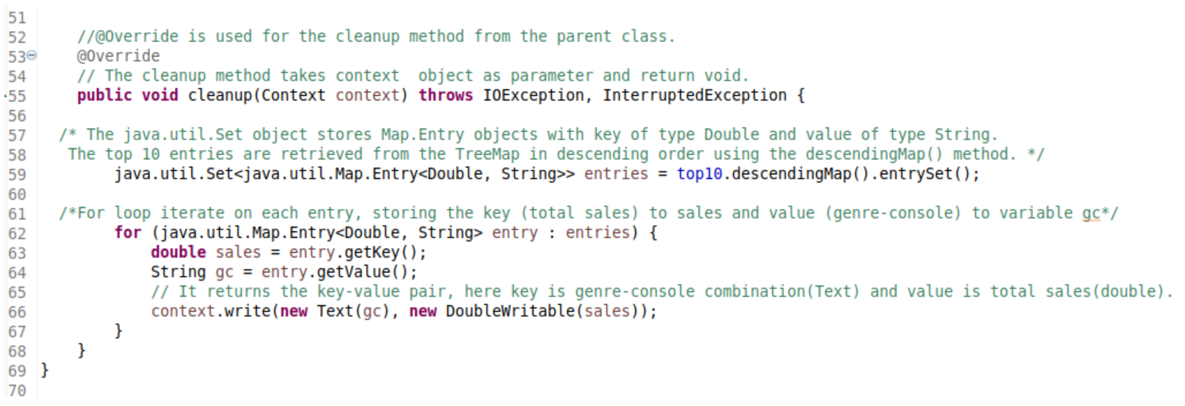
Reducer 2: TopNReducer


TopNReducer.java

```java
package org.myorg;

//Importing the libraries that contain the classes and methods needed in the Reducer class
import java.io.IOException;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;

/*Creating a subclass (child class) called TopNReducer from a parent
class called Reducer using the keyword extends */
public class TopNReducer extends Reducer<Text, NullWritable, Text, DoubleWritable> {

    /*Initialize integer variable TOP with value 10 which is used to keep track of the number of top genre-console combinations to be output.*/
    private static final int TOP= 10;

    /*Initialize private instance variable top10 which is a TreeMap object that maps sales to their corresponding genre-console combinations.*/
    private java.util.TreeMap<Double, String> top10;

    /*@override is used to setup method from the parent class.*/
    @Override

   /*initialize the topN instance variable with a new TreeMap object. This method is called once at the start of the reduce task.*/
    public void setup(Context context) throws IOException, InterruptedException {
        top10 = new java.util.TreeMap<>();
    }

```



```java
    /*@override is used to change the original behavior of the method “reduce” in the
    parent Class “Reducer” */
    @Override

    // the reduce method takes three parameters that are a key, a value, and a context as input parameters
    public void reduce(Text key, Iterable<NullWritable> values, Context context) throws IOException, InterruptedException {

        /*create a variable line of data type String, convert value into string and store it in line and
        Split the input text line using a tab("\t") as separator and store the result in the 'line' array.*/
        String[] line = key.toString().split("\t");

        //Store the key that is genre-console combination into genreConsole variable
        String genreConsole = line[0];

      //Store the value that is total sales variable totalSales.
        double totalSales = Double.parseDouble(line[1]);

        // Add the current genre-console combination to the TreeMap
        top10.put(totalSales, genreConsole);

        // Keep only the top 10 entries in the TreeMap
        if (top10.size() > TOP) {
            top10.remove(top10.firstKey());
        }
    }
```



```
51
52     //@Override is used for the cleanup method from the parent class.
53     @Override
54     // The cleanup method takes context  object as parameter and return void.
55     public void cleanup(Context context) throws IOException, InterruptedException {
56
57   /* The java.util.Set object stores Map.Entry objects with key of type Double and value of type String.
58    The top 10 entries are retrieved from the TreeMap in descending order using the descendingMap() method. */
59         java.util.Set<java.util.Map.Entry<Double, String>> entries = top10.descendingMap().entrySet();
60
61   /*For loop iterate on each entry, storing the key (total sales) to sales and value (genre-console) to variable gc*/
62         for (java.util.Map.Entry<Double, String> entry : entries) {
63             double sales = entry.getKey();
64             String gc = entry.getValue();
65             // It returns the key-value pair, here key is genre-console combination(Text) and value is total sales(double).
66             context.write(new Text(gc), new DoubleWritable(sales));
67         }
68     }
69 }
70
```

Figure 9: TopNReducer

The TopNReducer class is responsible for the reduce phase. The input key is the genre-console combination, and the input values are NullWritable objects. There is a variable TOP_N set to 10 and another variable topN as a TreeMap object.

A **TreeMap** in java is a data structure that holds key-value pairs in sorted order either ascending or descending and includes efficient methods for locating, inserting, and deleting entries. It is implemented as a Red-Black Tree, which allows it to maintain element order even after element insertion and deletion. The most significant advantages of utilising a TreeMap is that it guarantees logarithmic time complexity for most operations such as adding, removing, and searching items.

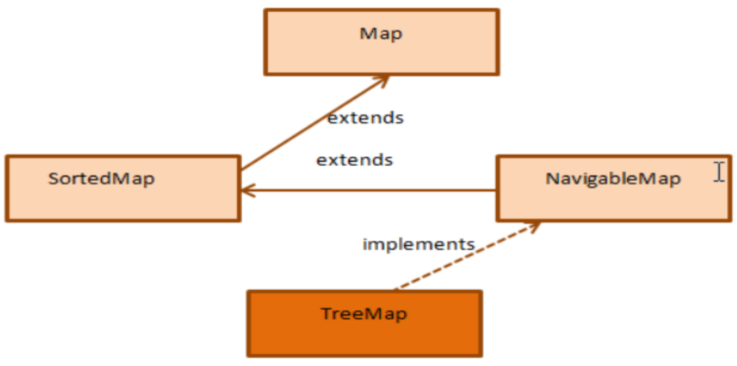
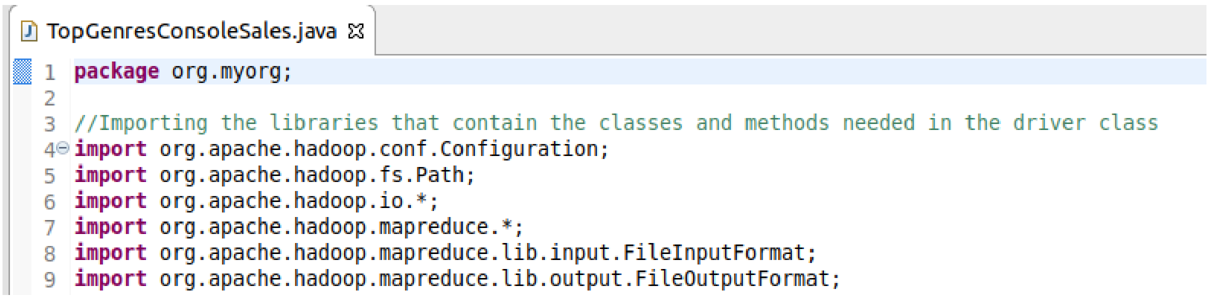


Figure 10: TreeMap Flow

In reducer the TreeMap is utilised to store genre-console combinations as keys and their associated sales as values. The line 47 of code is a if statement where the size of TreeMap is checked with each new entry and if the size exceeds 10, the lowest entry is removed.

In the cleanup method entries are retrieved from the TreeMap in descending order using the descendingMap() method as done in line 60, finally a for loop is used to iterate through the top 10 entries, storing the genre-console combination as a Text object and the total sales as a DoubleWritable object. The context object is then used to emit the key-value pairs as output records.

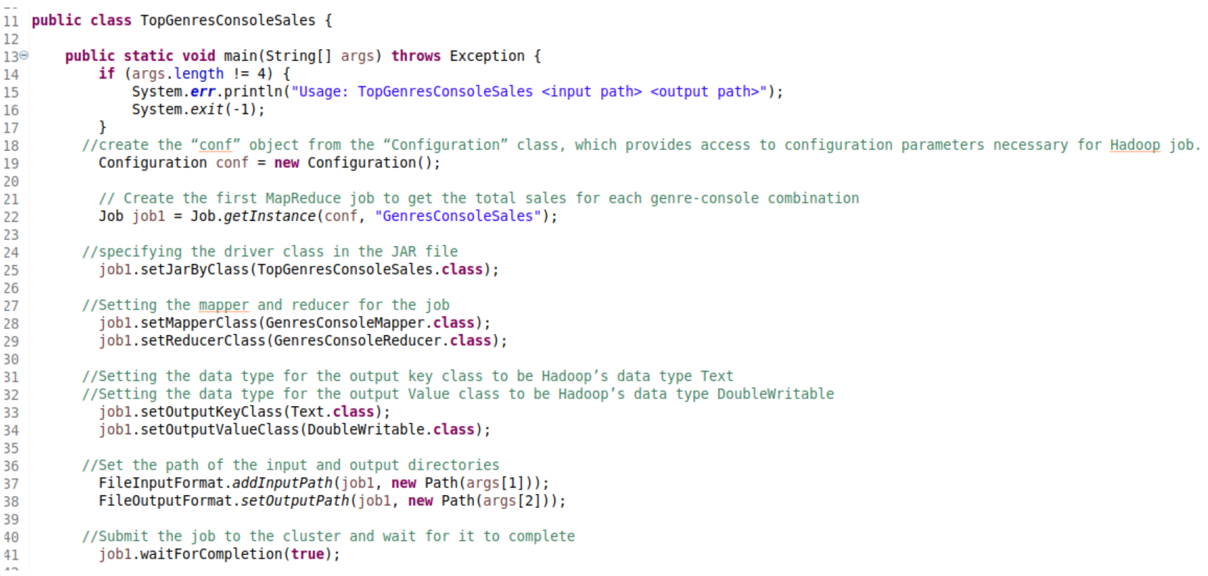
Main Driver Class: TopGenresConsoleSales


TopGenresConsoleSales.java

```
1 package org.myorg;
2
3 //Importing the libraries that contain the classes and methods needed in the driver class
4 import org.apache.hadoop.conf.Configuration;
5 import org.apache.hadoop.fs.Path;
6 import org.apache.hadoop.io.*;
7 import org.apache.hadoop.mapreduce.*;
8 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
9 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
```

Figure 11: Libraries of Driver Class

Despite the idea that the mapper and reducer representations are all that is needed to conduct a MapReduce job, MapReduce requires one additional piece of code: the driver class, which interfaces with the Hadoop framework and specifies the configuration parameters needed to run a MapReduce job. This includes instructing Hadoop on which Mapper and Reducer classes to use, where to get input data and in what format to find it, where to store output data, and how to organise and arrange it.

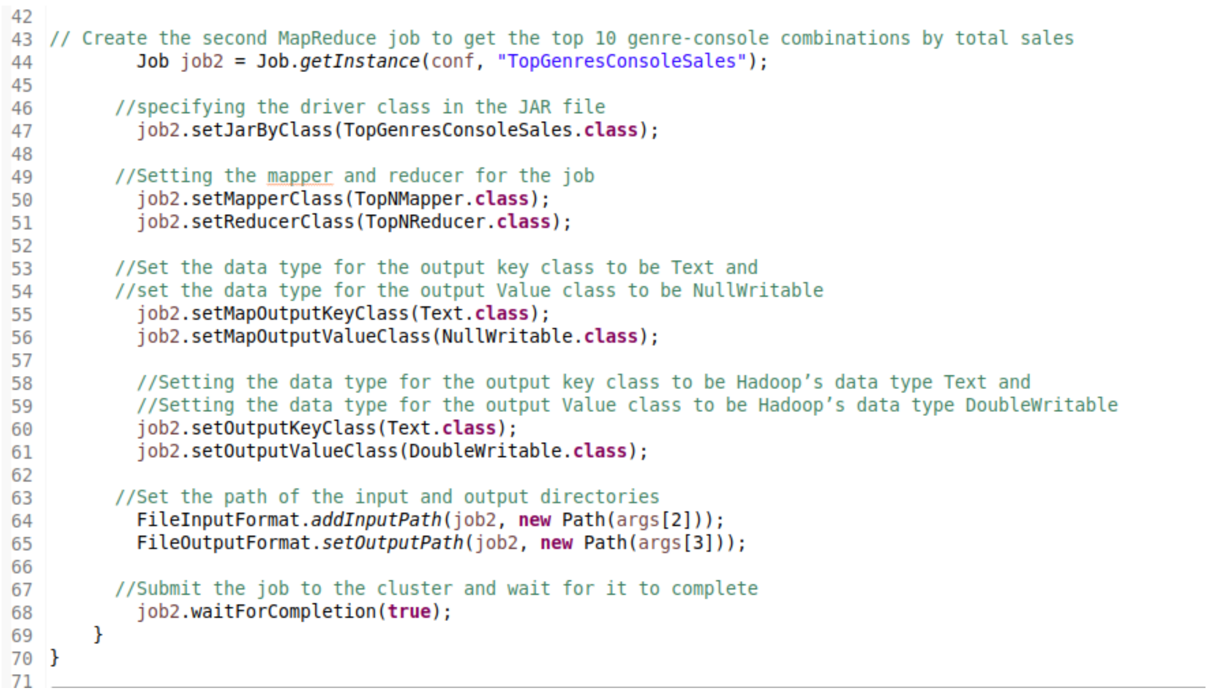


```
11 public class TopGenresConsoleSales {
12
13     public static void main(String[] args) throws Exception {
14         if (args.length != 4) {
15             System.err.println("Usage: TopGenresConsoleSales <input path> <output path>");
16             System.exit(-1);
17         }
18       //create the “conf” object from the “Configuration” class, which provides access to configuration parameters necessary for Hadoop job.
19         Configuration conf = new Configuration();
20
21         // Create the first MapReduce job to get the total sales for each genre-console combination
22         Job job1 = Job.getInstance(conf, "GenresConsoleSales");
23
24       //specifying the driver class in the JAR file
25         job1.setJarByClass(TopGenresConsoleSales.class);
26
27       //Setting the mapper and reducer for the job
28         job1.setMapperClass(GenresConsoleMapper.class);
29         job1.setReducerClass(GenresConsoleReducer.class);
30
31       //Setting the data type for the output key class to be Hadoop’s data type Text
32       //Setting the data type for the output Value class to be Hadoop’s data type DoubleWritable
33         job1.setOutputKeyClass(Text.class);
34         job1.setOutputValueClass(DoubleWritable.class);
35
36       //Set the path of the input and output directories
37         FileInputFormat.addInputPath(job1, new Path(args[1]));
38         FileOutputFormat.setOutputPath(job1, new Path(args[2]));
39
40       //Submit the job to the cluster and wait for it to complete
41         job1.waitForCompletion(true);
```

Figure 12: JOB 1 of Driver Class

In the above code line 19 is creating an object for the Configuration class which provides access to configuration parameters necessary for Hadoop job. The Job1 is created with the name “GenresConsoleSales”. The jar is set to the driver class (TopGenresConsoleSales) and the mapper and reducers are set to run the first MapReduce jobs namely GenresConsoleMapper and GenresConsoleReducer respectively. The setOutputKeyClass and setOutputValueClass are used to set the output key class to Text and the output value class to DoubleWritable.

FileInputFormat is used to specify the input and output folders. Finally, the task is sent to the Hadoop cluster using waitForCompletion, and the software waits for it to finish.



```
// Create the second MapReduce job to get the top 10 genre-console combinations by total sales
        Job job2 = Job.getInstance(conf, "TopGenresConsoleSales");

      //specifying the driver class in the JAR file
        job2.setJarByClass(TopGenresConsoleSales.class);

      //Setting the mapper and reducer for the job
        job2.setMapperClass(TopNMapper.class);
        job2.setReducerClass(TopNReducer.class);

      //Set the data type for the output key class to be Text and
      //set the data type for the output Value class to be NullWritable
        job2.setMapOutputKeyClass(Text.class);
        job2.setMapOutputValueClass(NullWritable.class);

        //Setting the data type for the output key class to be Hadoop’s data type Text and
        //Setting the data type for the output Value class to be Hadoop’s data type DoubleWritable
        job2.setOutputKeyClass(Text.class);
        job2.setOutputValueClass(DoubleWritable.class);

      //Set the path of the input and output directories
        FileInputFormat.addInputPath(job2, new Path(args[2]));
        FileOutputFormat.setOutputPath(job2, new Path(args[3]));

      //Submit the job to the cluster and wait for it to complete
        job2.waitForCompletion(true);
    }
}
```

Figure 13: JOB 2 of Driver Class

The Job2 is created with the name “GenresConsoleSales”. The jar is set to the driver class (TopGenresConsoleSales) and the mapper and reducers are set to run the first MapReduce jobs namely TopNMapper and TopNReducer respectively. The setOutputKeyClass and setOutputValueClass are used to set the output key class to Text and the output value class to DoubleWritable. FileInputFormat is used to specify the input and output folders. Finally, the task is sent to the Hadoop cluster using waitForCompletion, and the software waits for it to finish.

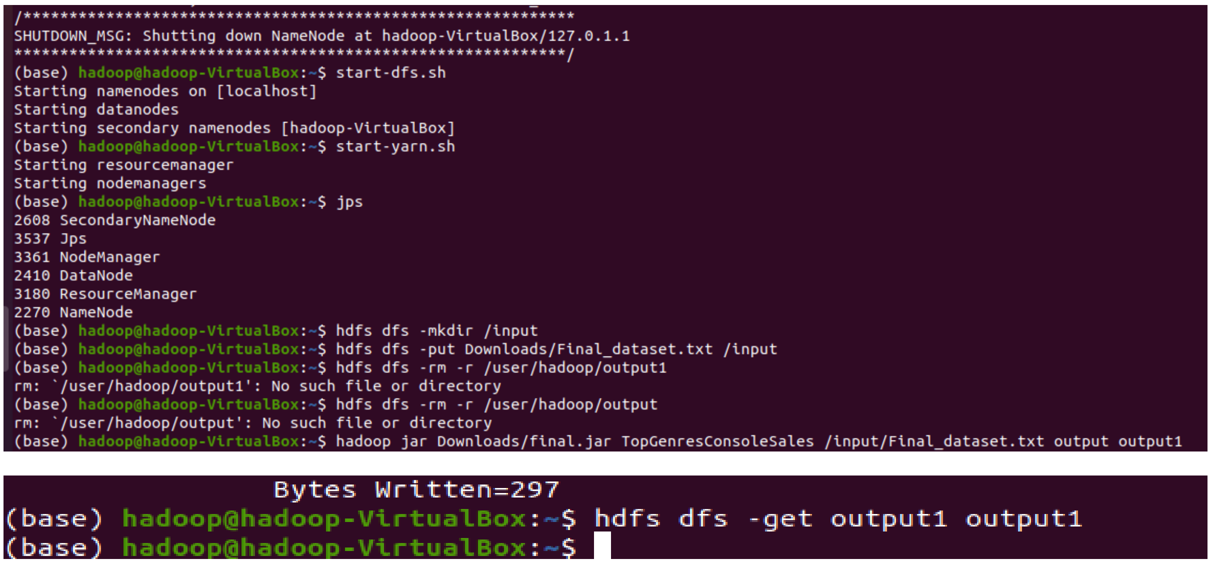
STAGE	INPUT	INPUT TYPE	OUTPUT	OUTPUT TYPE
Mapper 1 [GenresConsoleMapper]	Key: Input data file Location. Value: Line by Line Data	Key:LongWritable Value: Text	Key: Genre, Console. Value:Global_sales	Key: Text Value:DoubleWritable
Reducer 1 [GenresConsoleReducer]	Key: Genre, Console. Value: Global_sales	Key: Text Value:DoubleWritable	Key:Combination of Genre Console Value:Total Global_sales	Key: Text Value:DoubleWritable
Mapper 2 [TopNMapper]	Key: Output file of	Key:LongWritable Value: Text	Key:Line by Line data	Key: Text Value:NullWritable

	Reducer 1 Location. Value: Line by Line Data		Value:NULL	
Reducer 2 [TopNReducer]	Key: Genre, Console, Total Global_sales. Value: NULL	Key: Text Value:NullWritable	Key:Genre,Console Value:Total Global_sales	Key: Text Value:DoubleWritable

Table2: Description of Key-Value pair for each MapReduce

Result & Evaluation:

When the job is finished, a text file with the result is saved in the HDFS output directory. As a result, it must be transferred from the HDFS to the local system before it can be accessed.

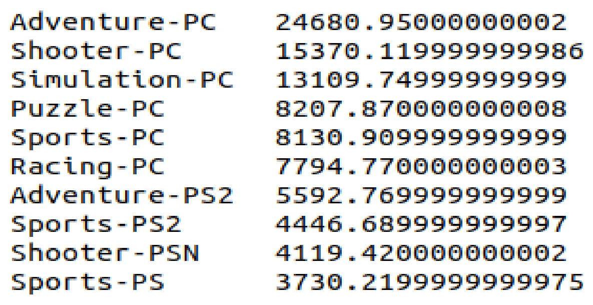


```
/************************************************************
SHUTDOWN_MSG: Shutting down NameNode at hadoop-VirtualBox/127.0.1.1
************************************************************/
(base) hadoop@hadoop-VirtualBox:~$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [hadoop-VirtualBox]
(base) hadoop@hadoop-VirtualBox:~$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
(base) hadoop@hadoop-VirtualBox:~$ jps
2608 SecondaryNameNode
3537 Jps
3361 NodeManager
2410 DataNode
3180 ResourceManager
2270 NameNode
(base) hadoop@hadoop-VirtualBox:~$ hdfs dfs -mkdir /input
(base) hadoop@hadoop-VirtualBox:~$ hdfs dfs -put Downloads/Final_dataset.txt /input
(base) hadoop@hadoop-VirtualBox:~$ hdfs dfs -rm -r /user/hadoop/output1
rm: `/user/hadoop/output1': No such file or directory
(base) hadoop@hadoop-VirtualBox:~$ hdfs dfs -rm -r /user/hadoop/output
rm: `/user/hadoop/output': No such file or directory
(base) hadoop@hadoop-VirtualBox:~$ hadoop jar Downloads/final.jar TopGenresConsoleSales /input/Final_dataset.txt output output1
```

```
                Bytes Written=297
(base) hadoop@hadoop-VirtualBox:~$ hdfs dfs -get output1 output1
(base) hadoop@hadoop-VirtualBox:~$
```

Figure 14: Setup the environment & run the program in Hadoop

The Figure 13 clearly mention every command that needs to be executed so that we receive the output. When we execute the last command “ **hdfs dfs -get output1 output1**” then the output file which existed in Hadoop is copied on the local system for the user to view. The below image is the final output for my execution.



```
Adventure-PC     24680.95000000002
Shooter-PC       15370.119999999986
Simulation-PC    13109.74999999999
Puzzle-PC        8207.870000000008
Sports-PC        8130.909999999999
Racing-PC        7794.770000000003
Adventure-PS2    5592.769999999999
Sports-PS2       4446.689999999997
Shooter-PSN      4119.420000000002
Sports-PS        3730.2199999999975
```

Figure 15: Final Output – Top 10 Genre Console combination with respect to total sales

Recall the research question: ***What are the top 10 genre console combinations that are popular among buyers based on total sales?***

To answer this question, we started with tones of data and using the jobs first figured out the different combinations of Genre and console and then their total sales. Furthermore, we analysed the long list of genre console combinations and then using another job came down to top 10 combinations. This analysis is helpful to answer the question, it's clear that the most popular combination among the buyers is Adventure and PC with 24680.95 as total sales. The top 10 combinations are displayed as output which clearly indicates the interest of the buyers.

To better evaluate these results, some simple informative graphics can be produced with Tableau:

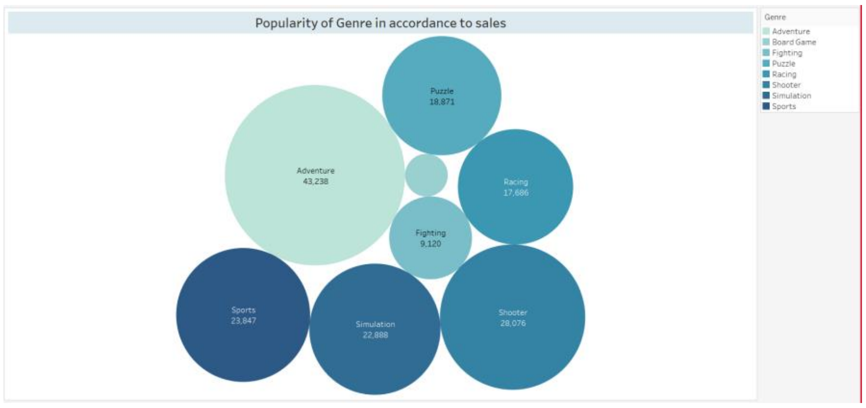


Figure 16: Popularity of genre with respect to sales

The above visualizations depict the relation of different genre and total global sales.

The bubble chart highlights the popularity of the 8-genre based on the total sales, bigger the bubble greater the contribution in the sales. From this we can easily identify that Adventure is the most popular among buyers.

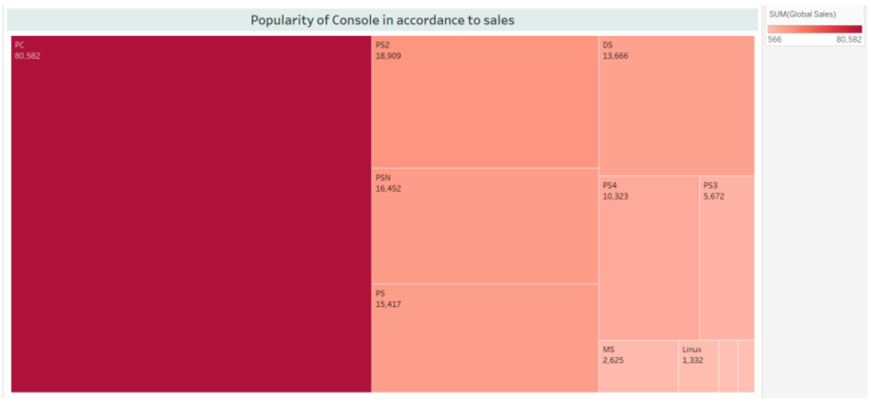


Figure 17: Popularity of console with respect to sales

The above visualizations depict the relation of different console and total global sales.
The TreeMap chart highlights the popularity of the 11-console based on the total sales, bigger the map area greater the contribution in the sales. From this we can easily identify that PC has been major contributor and is the most popular among buyers.

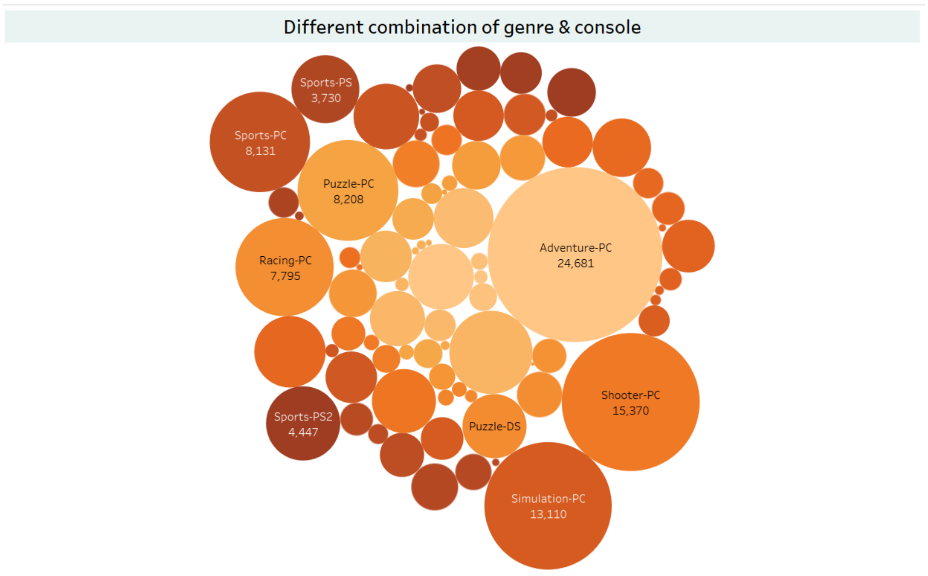


Figure 18: Visual Representation of the different combination of genre & console

From the first MapReduce job we identified that there are 81 combinations of genre and combinations, and their total global sales are calculated. The above bubble chart represents these combinations visually. The chart also highlights some of the top selling combinations like Adventure and PC (24,681), Shooter and PC (15,370) among others.

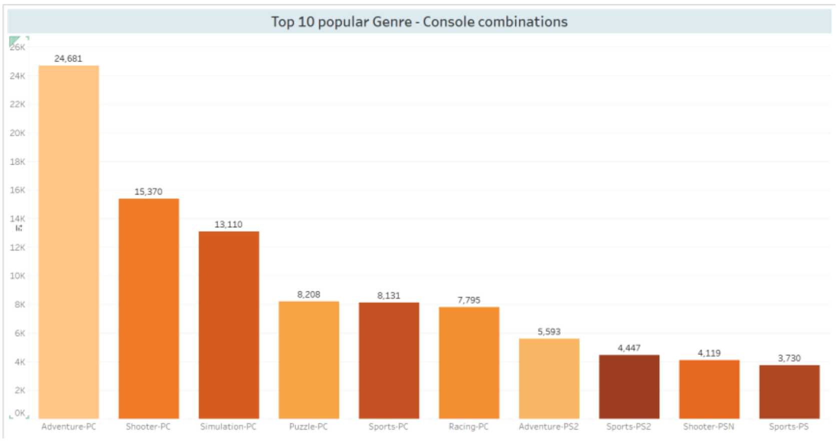


Figure 19: Visual Representation of MapReduce Job 2 (Top 10 combination based on sales)

This visualization depicts our outcome of MapReduce job that are the top 10 combinations which are most popular among buyers as their sales are maximum. Adventure and PC are the most popular combination among buyers with 24,681 million sales.

Critical Reflection:

During this project the knowledge on Hadoop MapReduce has been applied to answer a research question. Apart from the basic understanding of MapReduce new techniques were learnt to solve the problem in efficient way, like chained **MapReduce** and **TreeMap**.

To begin with two different MapReduce algorithms were used, one for finding total sales for different combinations of genre and console, and another one to perform sorting and providing top 10 combination with highest total sales. These two functionalities were executed sequentially one after another through chain MapReduce functionality.

The possible improvements or enhancements that can be made are as follows:

1. Run parallel MapReduce to calculate the average user score and combine it with current MapReduce to further narrow down the finding of the popularity of genre console based on user score.
2. In the current sequential MapReduce, we can add total number of shipment and use it along with the worldwide sales to find the revenue generated through the sales of different genre console combinations.