INFO 7250 Engineering of Big Data Systems
FINAL PROJECT REPORT

Kartik Kumar
NUID: 001057162

Dataset Details

Name: NBA Dataset
Description: This dataset was collected to work on NBA games data. It was made using the nba stats website to create this dataset (using rest api). It has data on games from 2003 - 2020.
Link: https://www.kaggle.com/nathanlauga/nba-games?select=teams.csv

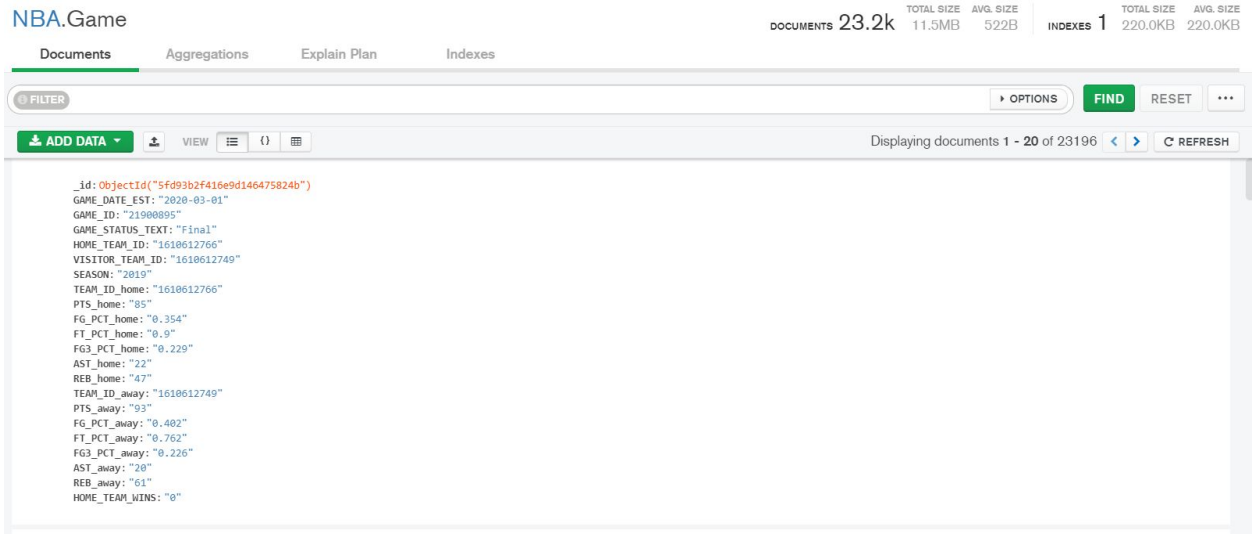
To showcase the complete analysis, I have categorized my analysis into 3 sections: Initial Exploratory analysis, Intermediate analysis, and Advanced analysis

1. Exploratory Analysis

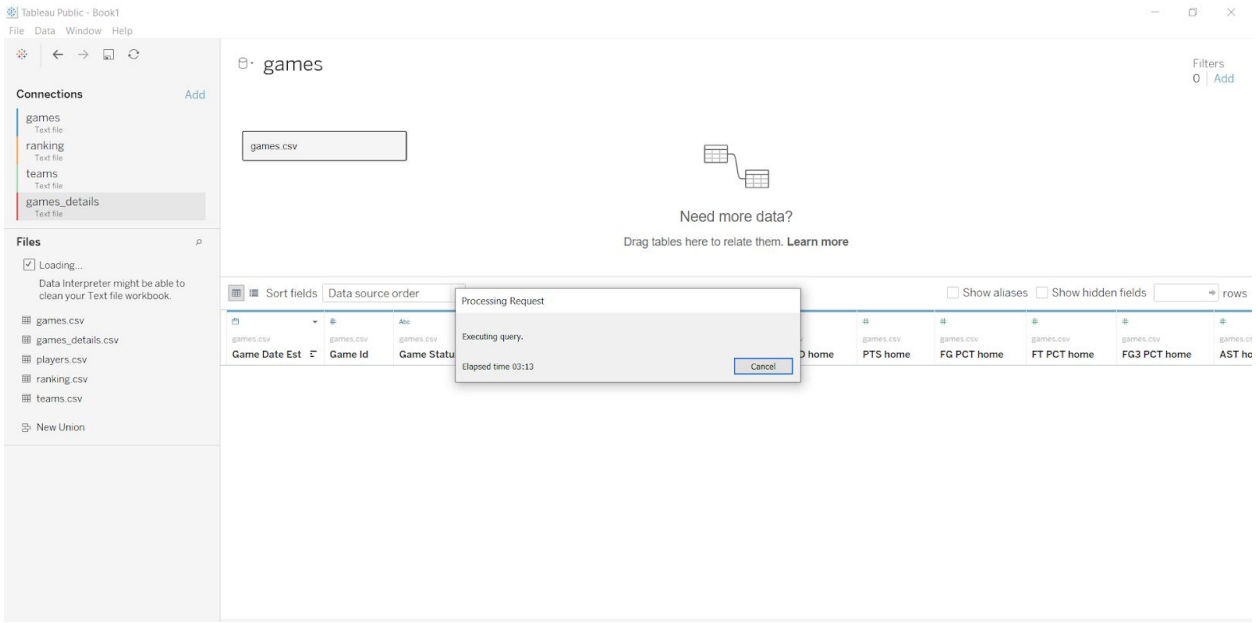
The data contains 5 csv files. I have done some basic exploration with each of the files below:

a) Games.csv

Here is a snippet of a single document/row from this file

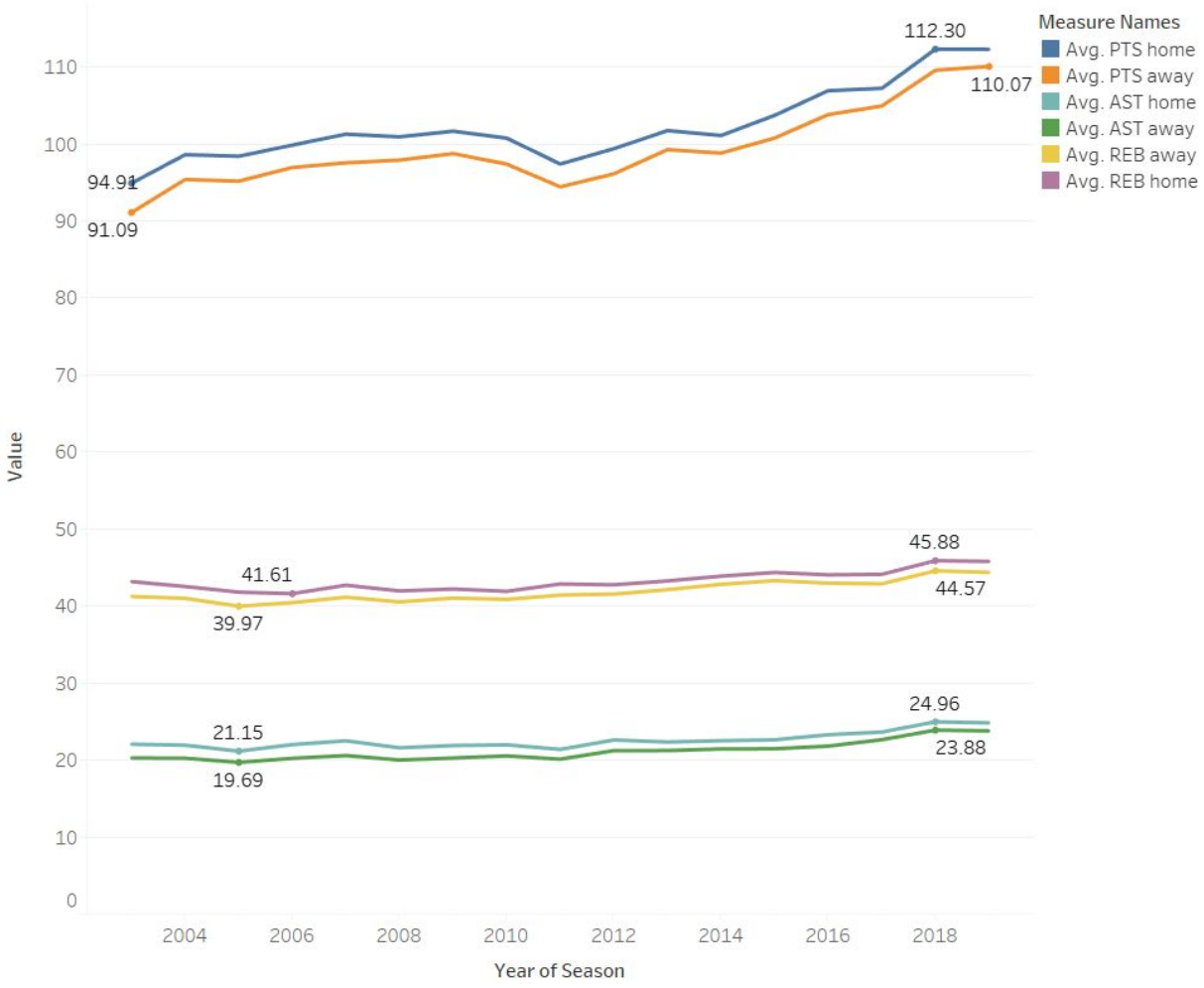


Using Tableau:
Automatic data cleaning by Data Interpreter



This feature enables the visualization tool to account for missing values in the data and automatically reads the data types of the columns.

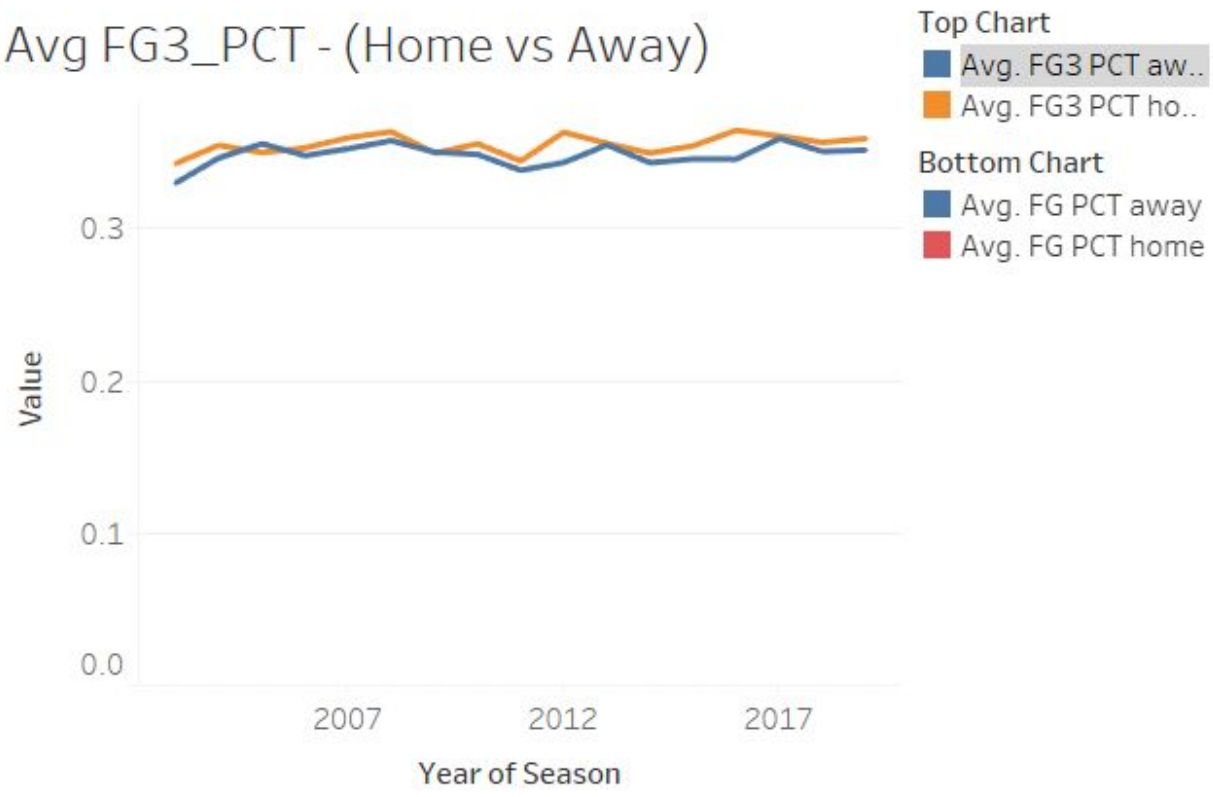
Visualization 1



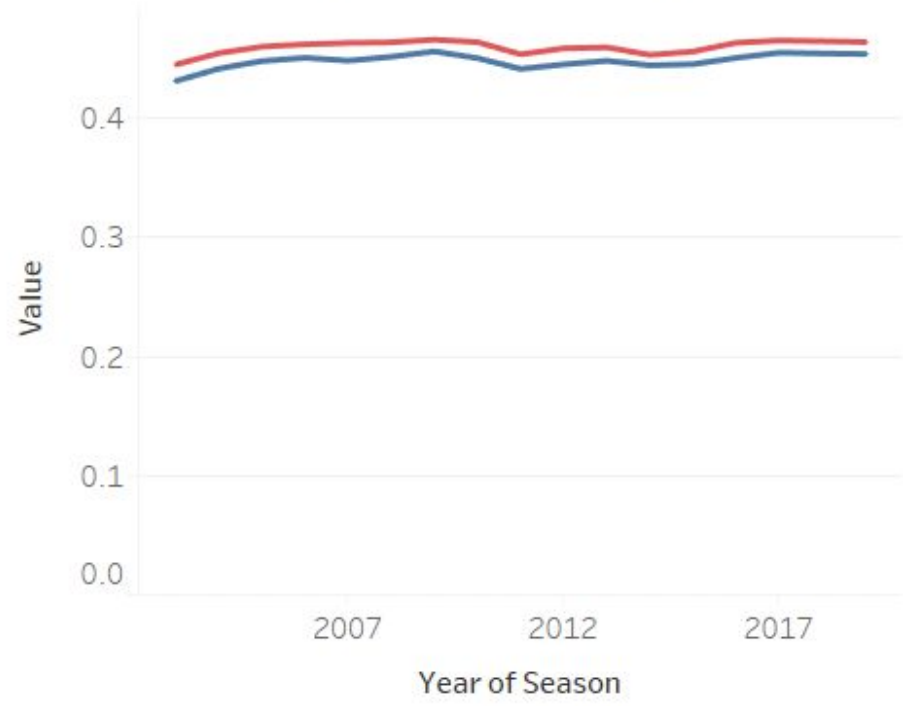
The above visualization shows that home teams have always had a better average than away teams in terms of total points per game, assists per game and rebounds per game.
Also, we can see that average points, assists and rebounds per game have increased in the last 17 years. The game is getting more aggressive and teams are understanding the importance of doing well in these statistics.
The markers show the all time Max and Min values for that line of data.
For e.g., Home teams had a lowest average of points per game in 2003 season =94.91
One would expect the highest to be the latest season, but the graph shows 2018 had the highest average home points per game.

Visualization 2:

Avg FG3_PCT - (Home vs Away)



Avg FG_PCT - (Home vs Away)



The above visualization shows that home teams have always had a higher shot conversion (fg_pct) than away teams. But in terms of 3 pointers, the away teams had a higher average in the year 2005 and 2009. Interesting!!

- Calculate the Games Per Season

Using MongoDB

```
> use NBA
switched to db NBA
> EDITOR = "Notepad"
Notepad
> var map = function(){}
> edit map
> var reduce = function(){}
> edit reduce
> edit map
> db.Game.mapReduce(map, reduce, {out:"GamesPerYear"})
{
        "result" : "GamesPerYear",
        "timeMillis" : 325,
        "counts" : {
                "input" : 23196,
                "emit" : 23196,
                "reduce" : 253,
                "output" : 19
        },
        "ok" : 1
}
> db.GamesPerYear.find()
{ "_id" : "", "value" : 1 }
{ "_id" : "2003", "value" : 564 }
{ "_id" : "2004", "value" : 1295 }
{ "_id" : "2005", "value" : 1435 }
{ "_id" : "2006", "value" : 1451 }
{ "_id" : "2007", "value" : 1404 }
{ "_id" : "2008", "value" : 1442 }
{ "_id" : "2009", "value" : 1428 }
{ "_id" : "2010", "value" : 1432 }
{ "_id" : "2011", "value" : 915 }
{ "_id" : "2012", "value" : 1580 }
{ "_id" : "2013", "value" : 1432 }
{ "_id" : "2014", "value" : 1441 }
{ "_id" : "2015", "value" : 1419 }
{ "_id" : "2016", "value" : 1429 }
{ "_id" : "2017", "value" : 1417 }
{ "_id" : "2018", "value" : 1382 }
{ "_id" : "2019", "value" : 1332 }
{ "_id" : "2020", "value" : 397 }
>
```

The above analysis shows the number of games played per year. For this analysis, I have used the column “GAME_DATE_EST” and split the date with year, month and date.

```
> var reduce = function(key, values){}
> edit reduce
> db.Game.mapReduce(map, reduce, {out:"SeasonCount"})
{
        "result" : "SeasonCount",
        "timeMillis" : 218,
        "counts" : {
                "input" : 23196,
                "emit" : 23196,
                "reduce" : 250,
                "output" : 18
        },
        "ok" : 1
}
> db.SeasonCount.find()
{ "_id" : null, "value" : 1 }
{ "_id" : "2003", "value" : 1385 }
{ "_id" : "2004", "value" : 1362 }
{ "_id" : "2005", "value" : 1432 }
{ "_id" : "2006", "value" : 1419 }
{ "_id" : "2007", "value" : 1411 }
{ "_id" : "2008", "value" : 1425 }
{ "_id" : "2009", "value" : 1424 }
{ "_id" : "2010", "value" : 1422 }
{ "_id" : "2011", "value" : 1104 }
{ "_id" : "2012", "value" : 1420 }
{ "_id" : "2013", "value" : 1427 }
{ "_id" : "2014", "value" : 1418 }
{ "_id" : "2015", "value" : 1416 }
{ "_id" : "2016", "value" : 1405 }
{ "_id" : "2017", "value" : 1382 }
{ "_id" : "2018", "value" : 1378 }
{ "_id" : "2019", "value" : 965 }
```

For the above analysis, I have used another column, which is the correct column to be used when analyzing games per year - “SEASON”. As seen from above, a Season may consist of two years. Typically, the league starts in October and ends in April month of the following year. The players have a 6 month off-season where they rest and prepare for the next season.

Time Taken by MongoDB = 218ms

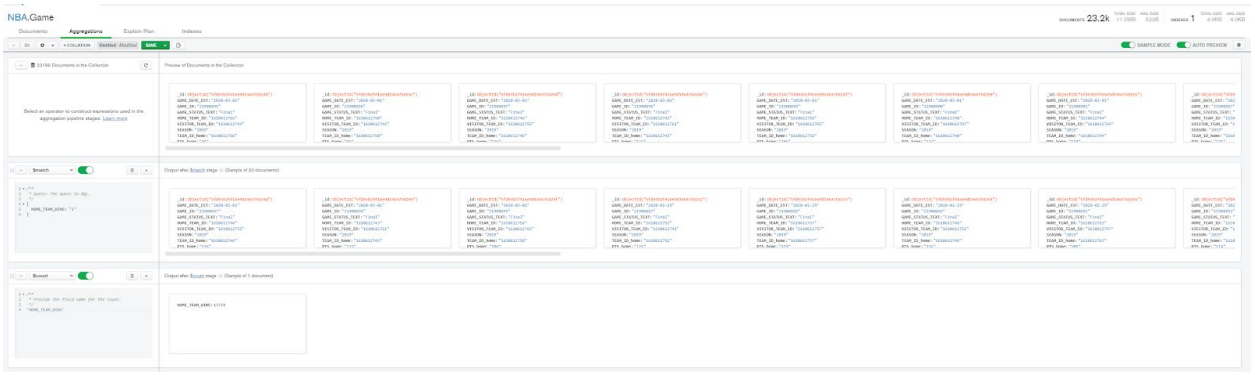
Using HDFS:

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -head /GamePerSeasonMR/part-r-00000
2020-12-15 12:35:24,225 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
2003    1385
2004    1362
2005    1432
2006    1419
2007    1411
2008    1425
2009    1424
2010    1422
2011    1104
2012    1420
2013    1427
2014    1418
2015    1416
2016    1405
2017    1382
2018    1378
2019    965
```

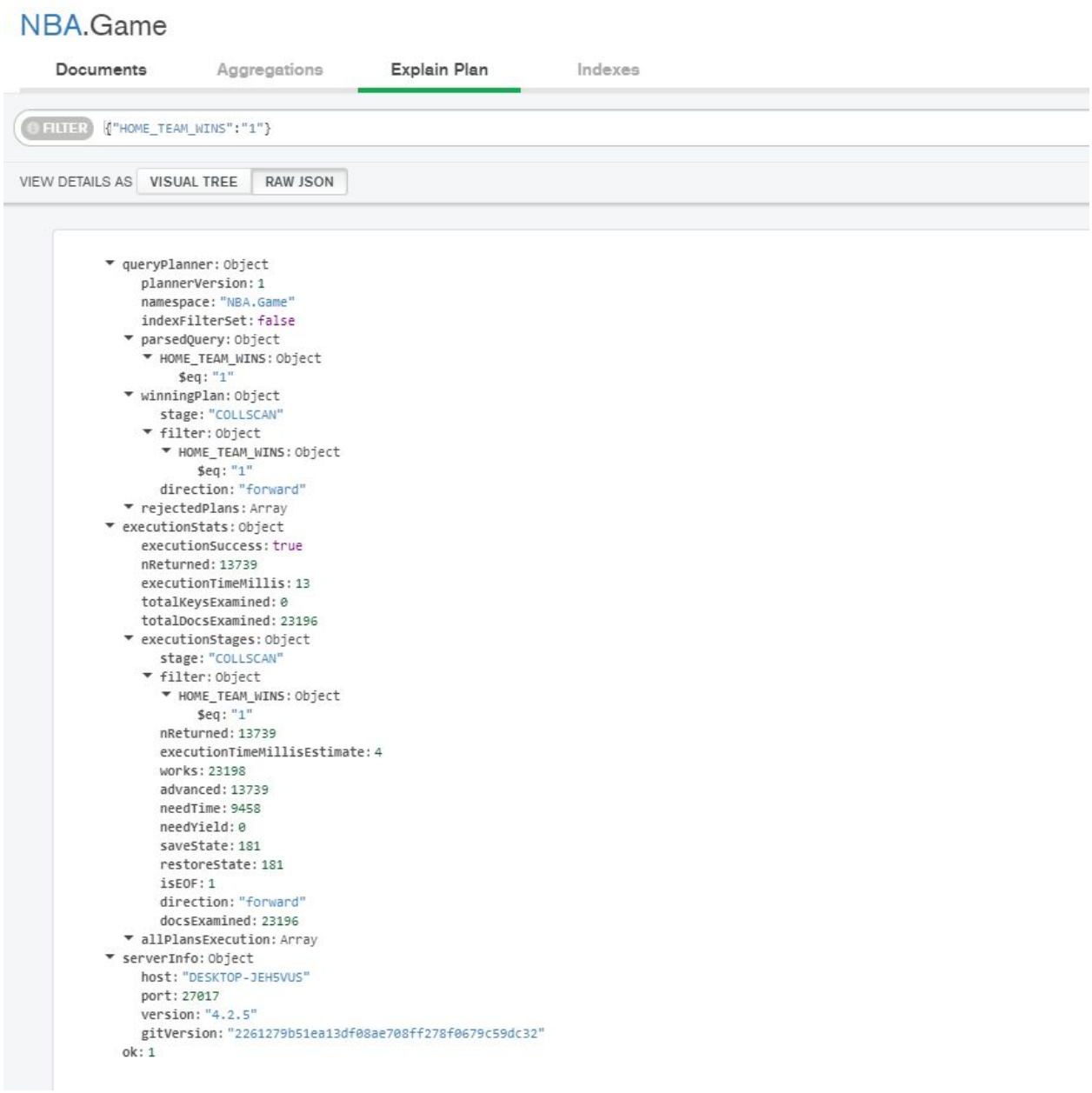
The above analysis was performed after loading the csv file into a single node hadoop cluster. As expected, the results are the same, but hadoop framework takes more time for this basic analysis.
Overall time taken by Hadoop = 15 seconds

- Home Team wins vs Count(Home_pts > away_pts)

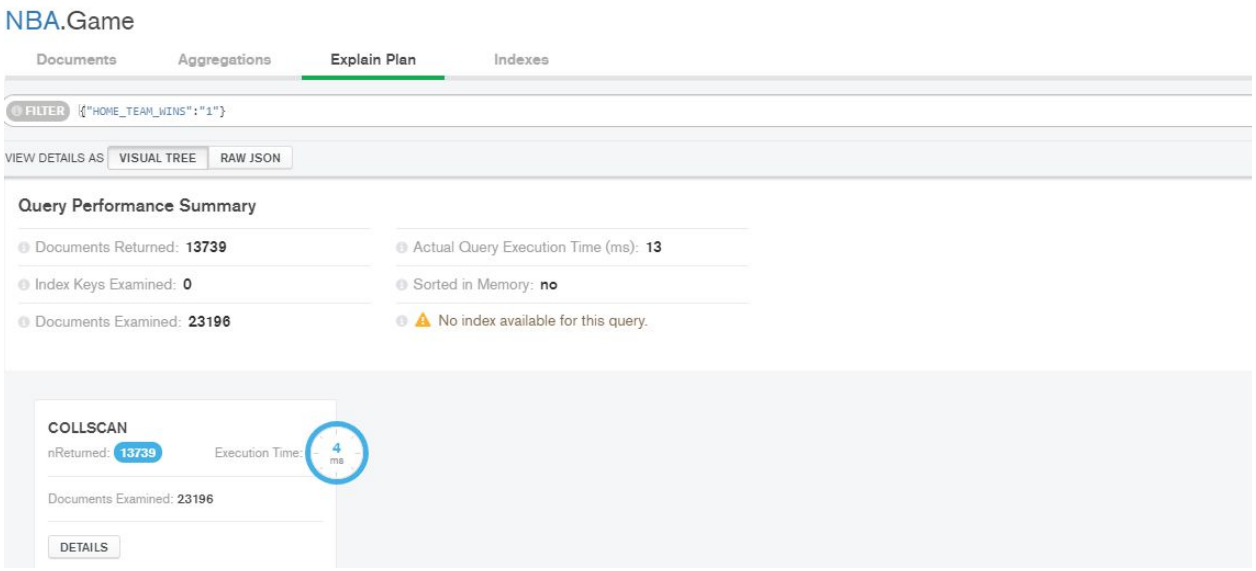
Using MongoDB:t
This is the aggregation dashboard of MongoDB Compass. It is a powerful tool as per the documentation and I thought of experimenting this capability of Compass.



In the following picture, we can see the details of any query run through Compass. For simplicity purpose, and since I was new to this, I experimented with a very basic query to calculate how many games did a home team win in the Games.csv file.



Note that this shows 13739 documents were returned out of the total 23198 when query ran on the database.



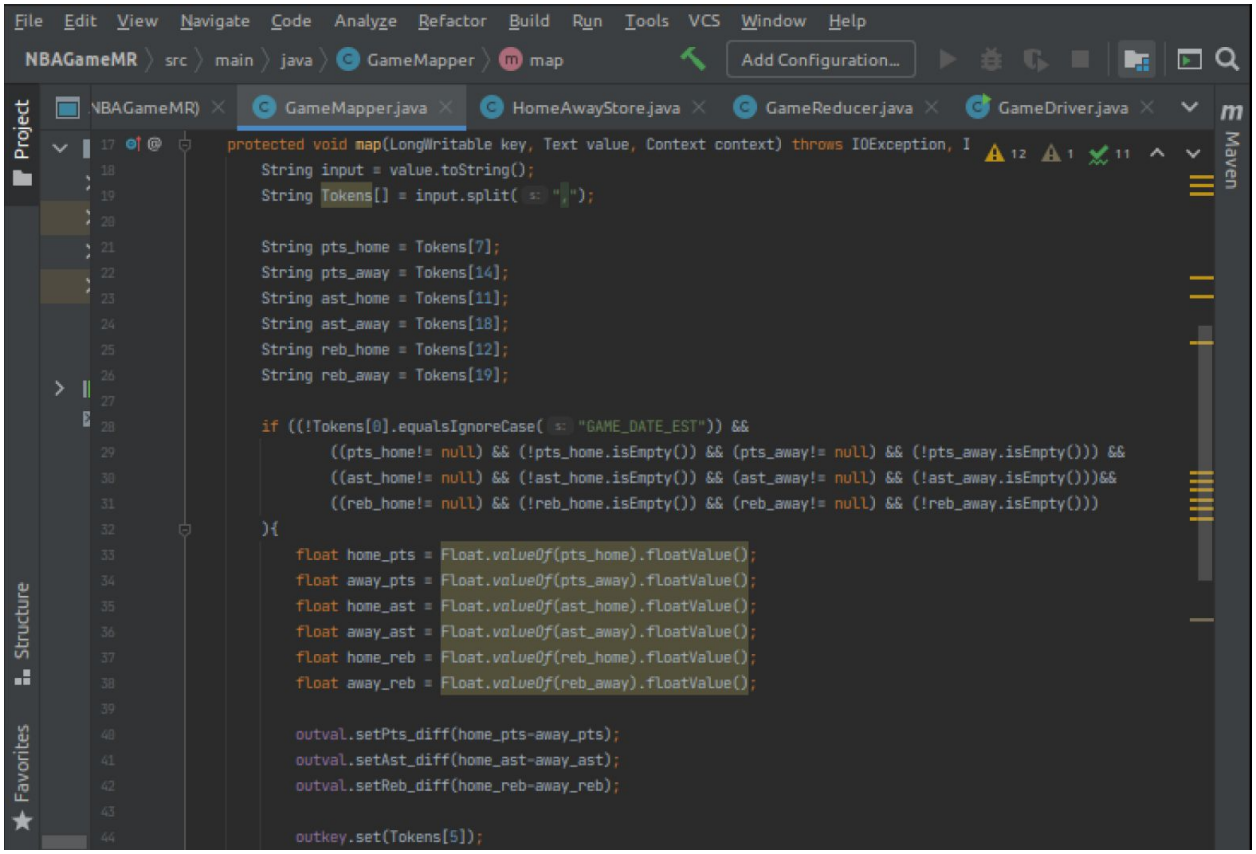
The dashboard allows us to see query performance in raw json format and in terms of a dashboard visualisation. Definitely need to try this out with more complex queries!

Using HDFS:
Overall Time Taken = 12 seconds

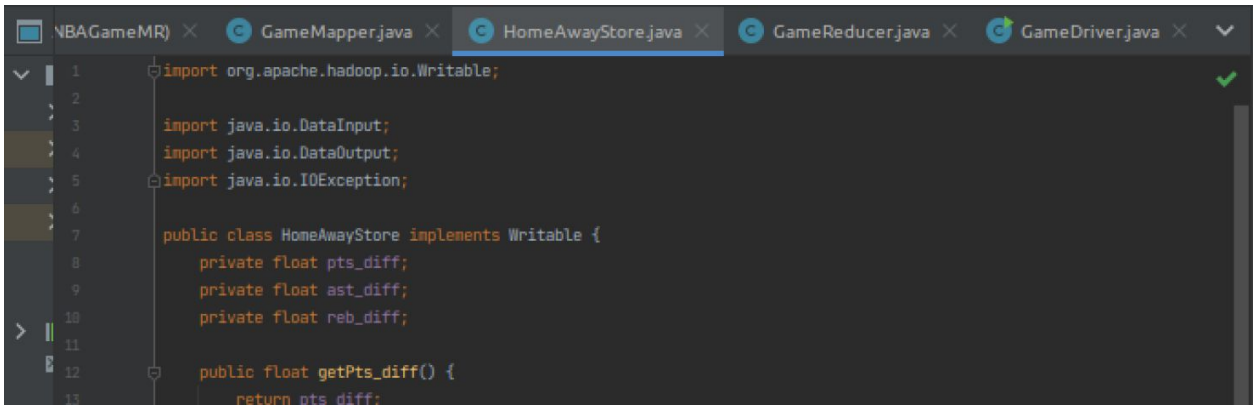
```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -head /HomeWinsCount/part-r-00000
2020-12-15 15:56:59,423 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
HomeWins        13739.0
```

A similar result to that seen in Compass. Home team has won 13739 games.

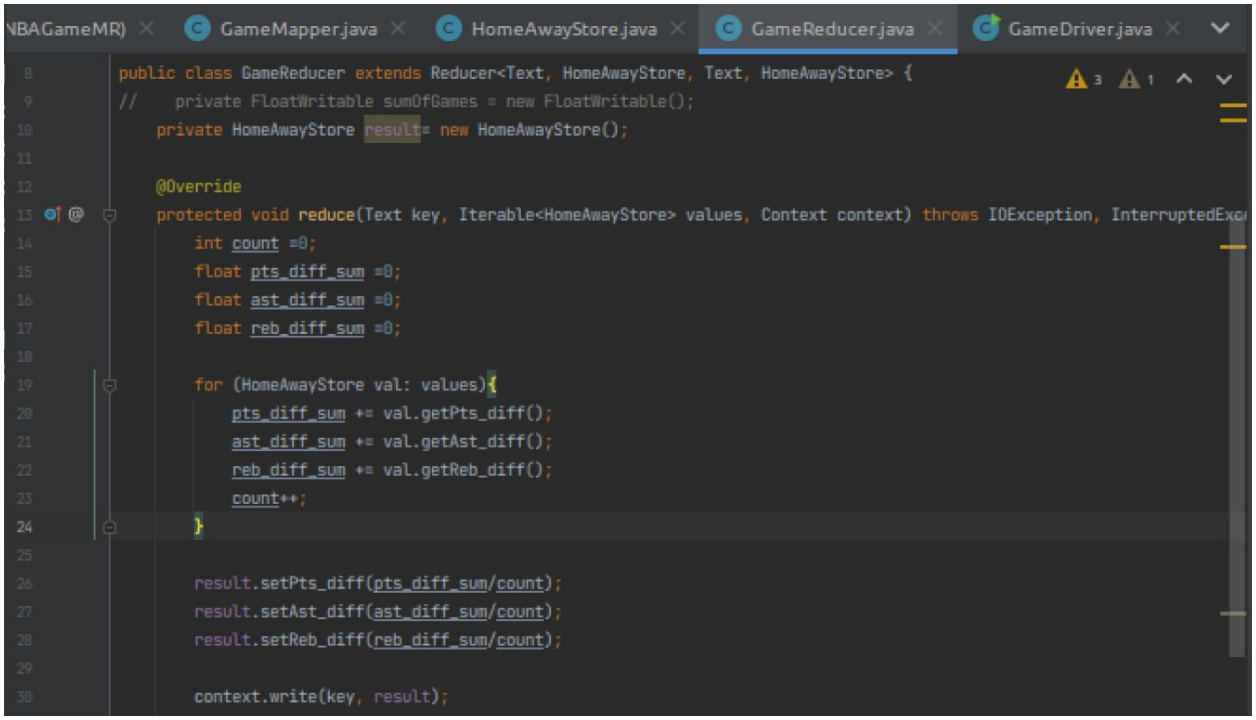
- Is there Home advantage?
  Using HDFS:
  Overall Time Taken = 20seconds
  Code Snippet:
    - Mapper - Same mapper file has been modified for all basic analysis on this csv file



- Custom Writable



- Reducer - Same reducer file has been modified for all basic analysis on this csv file




```java
public class GameReducer extends Reducer<Text, HomeAwayStore, Text, HomeAwayStore> {
//    private FloatWritable sumOfGames = new FloatWritable();
    private HomeAwayStore result= new HomeAwayStore();

    @Override
    protected void reduce(Text key, Iterable<HomeAwayStore> values, Context context) throws IOException, InterruptedExc
        int count =0;
        float pts_diff_sum =0;
        float ast_diff_sum =0;
        float reb_diff_sum =0;

        for (HomeAwayStore val: values){
            pts_diff_sum += val.getPts_diff();
            ast_diff_sum += val.getAst_diff();
            reb_diff_sum += val.getReb_diff();
            count++;
        }

        result.setPts_diff(pts_diff_sum/count);
        result.setAst_diff(ast_diff_sum/count);
        result.setReb_diff(reb_diff_sum/count);

        context.write(key, result);
```

- Driver - Same driver file has been modified for all basic analysis on this csv file

```java
public class GameDriver {
    public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {
        Configuration conf = new Configuration();
        FileSystem fs =FileSystem.get(conf);
        if(fs.exists(new Path(args[1])))
        {
            fs.delete(new Path(args[1]),  b: true);
        }
        Job job = Job.getInstance(conf);
        job.setMapperClass(GameMapper.class);
        job.setReducerClass(GameReducer.class);
//        job.setCombinerClass(GameReducer.class);
        job.setJarByClass(GameDriver.class);

        TextInputFormat.addInputPath(job, new Path(args[0]));
        TextOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setInputFormatClass(TextInputFormat.class);
        job.setMapOutputKeyClass(Text.class);
//        job.setMapOutputValueClass(FloatWritable.class);
        job.setMapOutputValueClass(HomeAwayStore.class);
        job.setOutputKeyClass(Text.class);
//        job.setOutputValueClass(FloatWritable.class);
        job.setOutputValueClass(HomeAwayStore.class);
        System.exit(job.waitForCompletion( verbose: true) ? 0 : 1);
    }
}
```

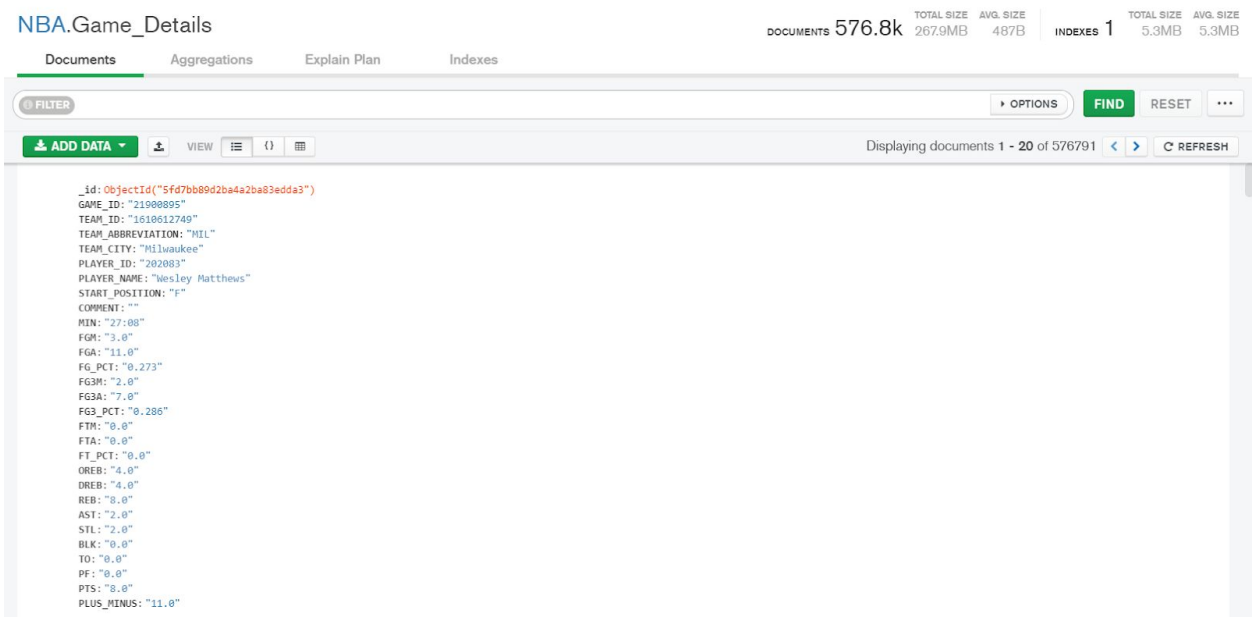
Output:

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -head /AvgDiffPerSeason/part-r-00000
2020-12-15 17:24:40,678 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
2003    HomeAwayStore{pts_diff=3.81493, ast_diff=1.7947123, reb_diff=1.9323484}
2004    HomeAwayStore{pts_diff=3.2232013, ast_diff=1.688693, reb_diff=1.5330397}
2005    HomeAwayStore{pts_diff=3.2339385, ast_diff=1.4622905, reb_diff=1.8275139}
2006    HomeAwayStore{pts_diff=2.9048626, ast_diff=1.7815363, reb_diff=1.167019}
2007    HomeAwayStore{pts_diff=3.724309, ast_diff=1.9177887, reb_diff=1.5556343}
2008    HomeAwayStore{pts_diff=3.0189474, ast_diff=1.5859649, reb_diff=1.4294736}
2009    HomeAwayStore{pts_diff=2.9087079, ast_diff=1.625, reb_diff=1.1748595}
2010    HomeAwayStore{pts_diff=3.3720112, ast_diff=1.443038, reb_diff=1.04782}
2011    HomeAwayStore{pts_diff=2.9628623, ast_diff=1.2653985, reb_diff=1.4393116}
2012    HomeAwayStore{pts_diff=3.2676055, ast_diff=1.387324, reb_diff=1.2070422}
2013    HomeAwayStore{pts_diff=2.473721, ast_diff=1.1072179, reb_diff=1.1359495}
2014    HomeAwayStore{pts_diff=2.2799718, ast_diff=1.0712271, reb_diff=1.0655854}
2015    HomeAwayStore{pts_diff=2.94774, ast_diff=1.1652542, reb_diff=1.0642655}
```

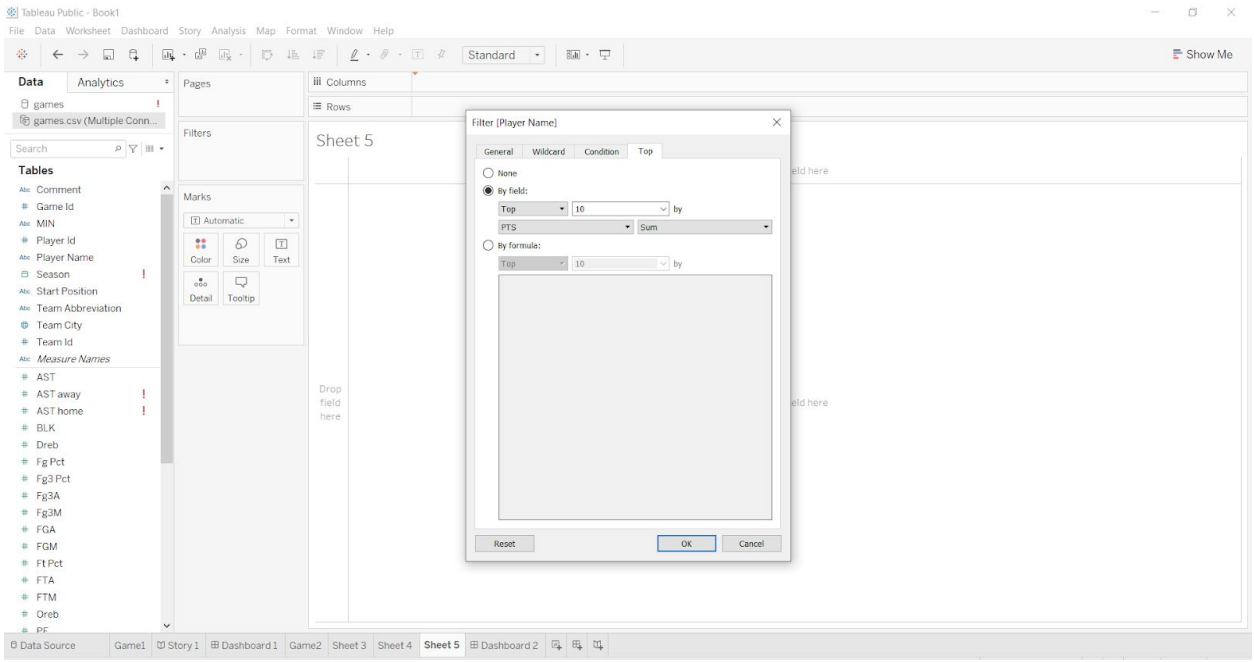
As seen in the output, in all seasons, home team has scored more than away and has had more assists and rebounds per game. Similar results to our analysis via Tableau!

2. Game Details.csv:

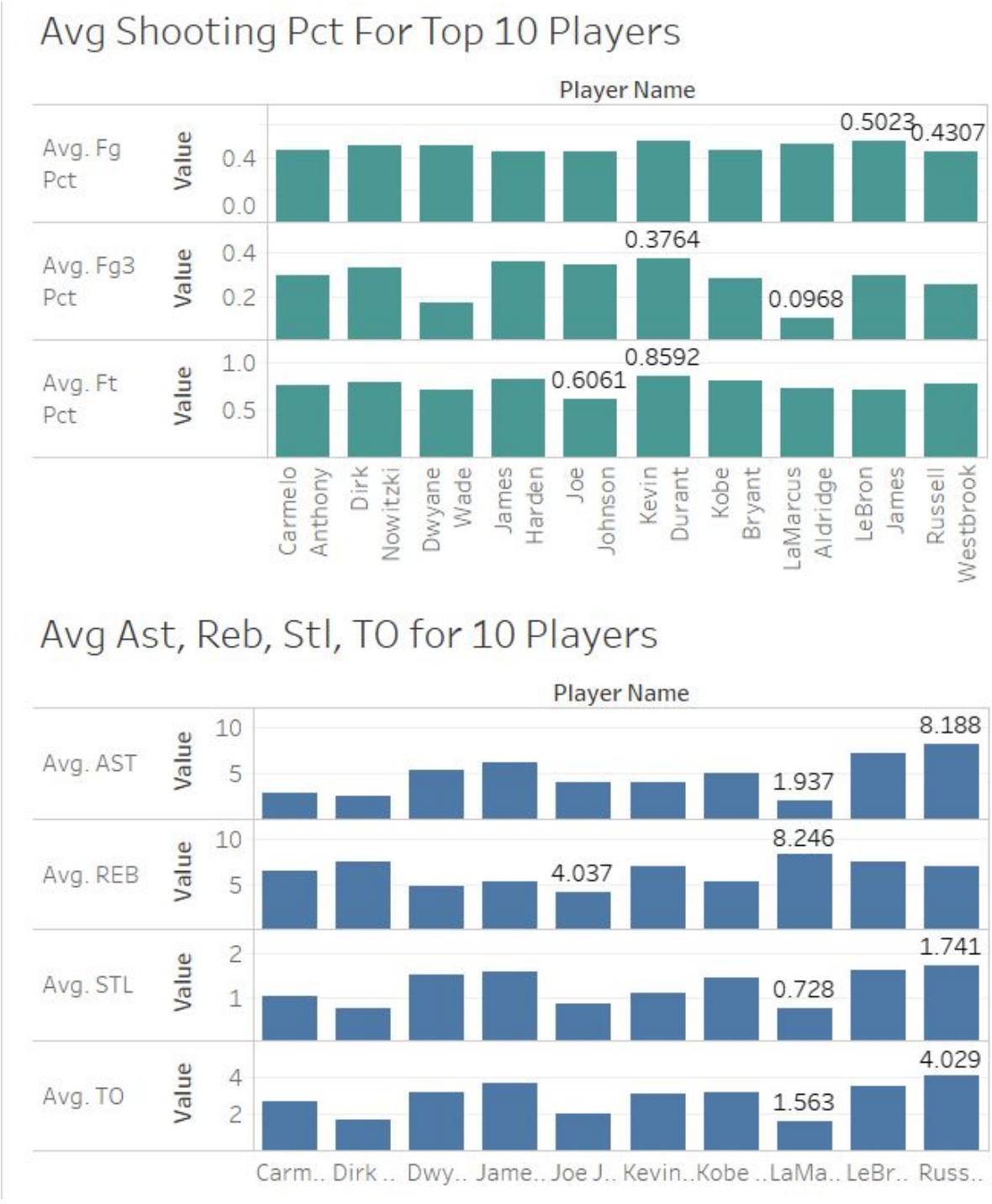
Moving on to analysing our second csv file a bit. Here is a snippet of a single document/row from this file.



Using Tableau:
For visualization purposes, since this data has the record of all NBA player who played in a game, I have restricted visualization to only the top 10 players. The field to decide the 10 players is the sum of Points. Here is how to add this filter in the visualization:

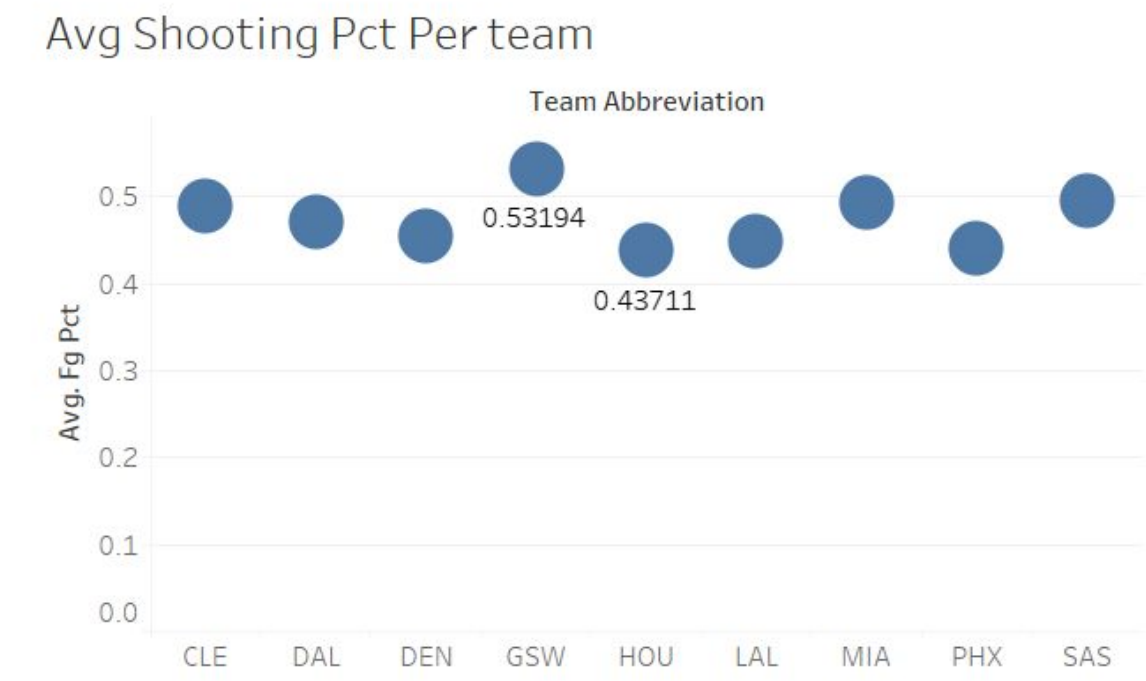


Visualization 1-



The above visualization shows the Top 10 players in terms of points scored and compares their avg statistics per game. The minimum and maximum averages have been labeled. For e.g., Russel Westbrook has the highest average assists per game and the lowest field goal percentage (FG_PCT). No wonder he passes the ball more than the others!

Visualization 2 -



Golden State Warriors have the highest Field Goal Percentage. Definitely the reason could be the splash brothers - Steph Curry & Klay Thompson. They helped the team win overall championship in 2015, 2017 & 2018.

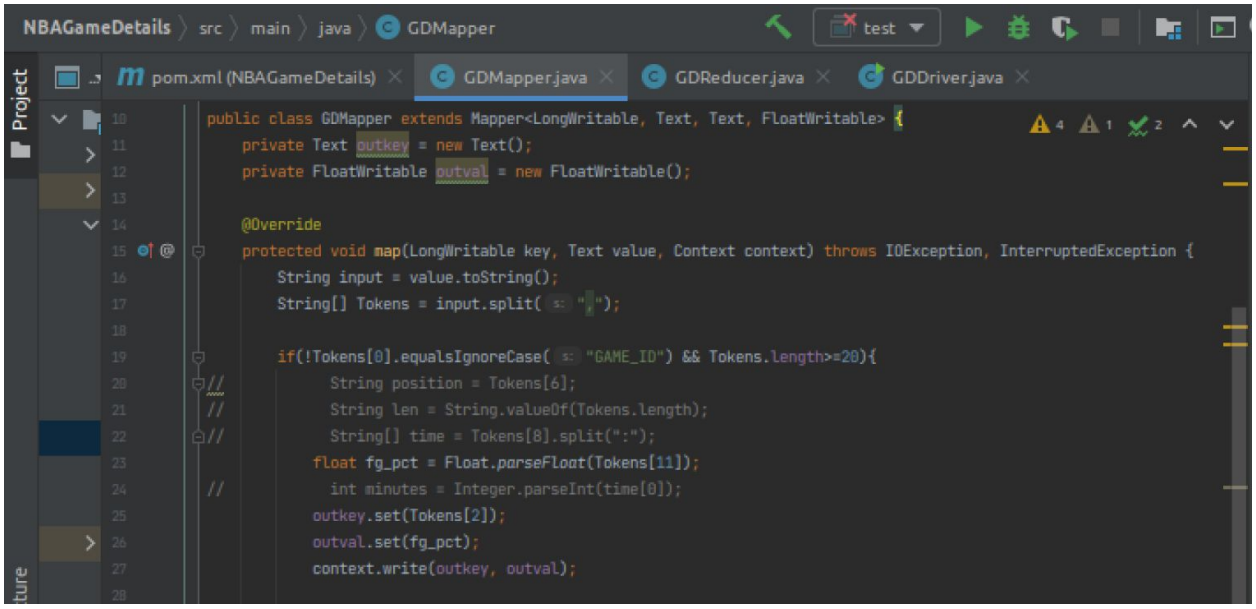
a) Players per team

Using HDFS:

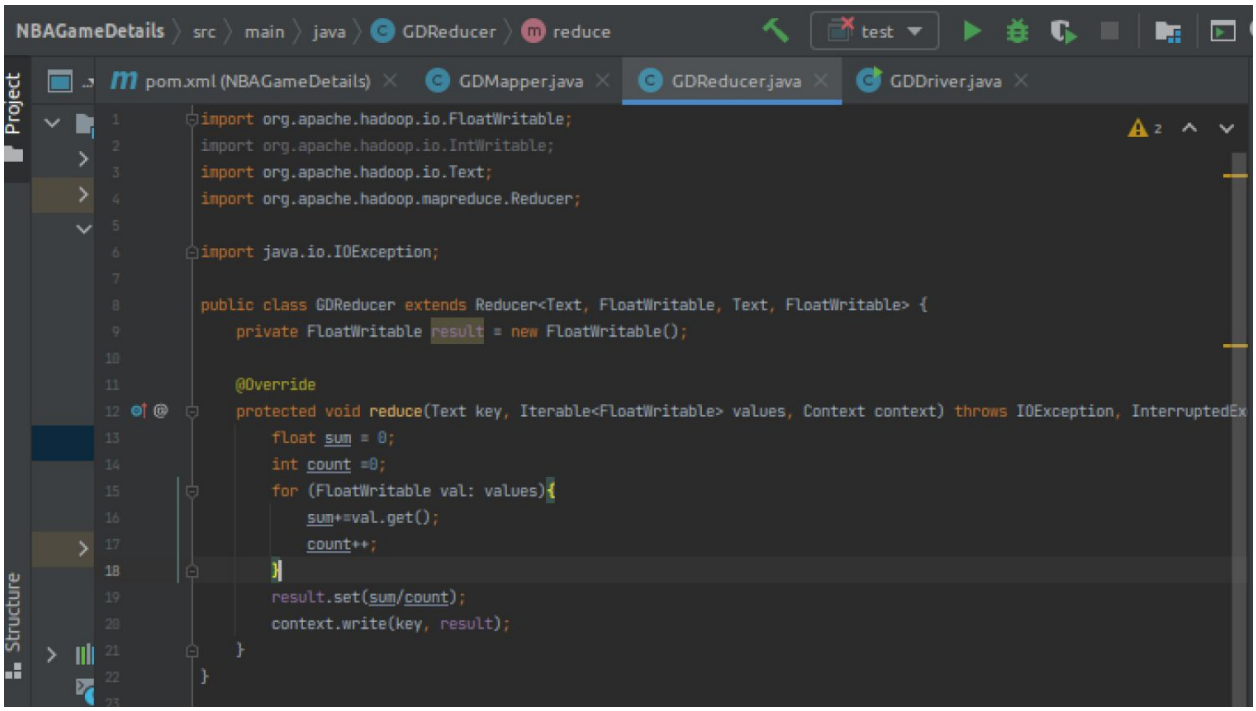
Overall Time Taken = 14 sec

Code Snippet:

- Mapper - Same mapper file has been modified for all basic analysis on this csv file



- Reducer - Same reducer file has been modified for all basic analysis on this csv file



Output:

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -head /PlayersPerTeamOverall/part-r-00000
2020-12-15 19:26:49,357 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
ATL     19335
BKN     9095
BOS     20199
CHA     17625
CHI     19079
CLE     19894
DAL     19714
DEN     19168
DET     19743
GSW     19742
HOU     19347
IND     19666
LAC     19024
LAL     19974
MEM     18978
MIA     20480
MIL     18911
MIN     18436
NJN     9765
NOH     8669
NOK     2137
NOP     7961
NYK     18337
OKC     14131
ORL     19056
PHI     18649
PHX     18837
POR     19039
SAC     18400
SAS     20727
SEA     5314
TOR     19241
UTA     19181
WAS     18928
```

Some of the teams that have a count of less than 12000 have that because the team either got merged with another team or the city no longer has a team. E.g. New Jersey Nets - renamed to Brooklyn Nets.

b)Players per position

Another example of counting the players w.r.t their Starting position in the game. There are 3 types:

C- Center

F - Forward

G - Guard

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -head /PlayersPerPosition/part-r-00000
2020-12-15 21:48:36,574 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
        355407
C       44274
F       88551
G       88549
```

As a part of generating the above analysis, I got to understand the missing values in the data. Below is a snippet of a mapreduce job I used to determine the Token Length - Token is the array of values made by splitting a line with “,”.

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -head /TokenLength/part-r-00000
2020-12-15 21:41:14,189 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
27      23661
28      460860
6       1
8       91819
9       441
```

Many rows did not have all 28 column values. As shown above, some rows only had 8-9 values.

c)Avg Minutes Per Game

Below I have used HDFS Mapreduce to compute the average minutes played by a player. The key emitted by the Mapper was the Player Name and the value was a float value of the minutes played in a game. To extract the Minutes played, the token was split on the ":" as the time had minutes and seconds.

The average calculation was done on the reducer side and since average is not a commutative and associative operation, hence no combiner was used.

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -tail /MinutesPerPlayer/part-r-00000
2020-12-15 22:04:45,562 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
wundu   16.0
Wesley Johnson  21.0
Wesley Matthews 31.0
Wesley Person   16.0
Wesley Saunders 3.0
Wesley Witherspoon      20.0
Will Barton     23.0
Will Blalock    10.0
Will Bynum      18.0
Will Cherry     9.0
Will Conroy     11.0
Will Cummings   12.0
Will Solomon    13.0
William Howard  8.0
Willie Cauley-Stein     23.0
Willie Deane    5.0
Willie Green    19.0
Willie Reed     11.0
Willie Warren   6.0
Willy Hernangomez       13.0
Wilson Chandler 28.0
Xavier Henry    15.0
Xavier Munford  13.0
Xavier Rathan-Mayes     17.0
Xavier Silas    8.0
Yakhouba Diawara        13.0
Yakuba Ouattara 3.0
Yante Maten     9.0
Yanyuhang Ding  8.0
Yao Ming        32.0
Yaroslav Korolev        7.0
Yi Jianlian     21.0
Yogi Ferrell    20.0
Youssou Ndoye   9.0
Yudai Baba      7.0
Yuta Tabuse     8.0
```

```
Yuta Tabuse     8.0
Yuta Watanabe   8.0
Zabian Dowdell  11.0
Zach Collins    17.0
Zach LaVine     29.0
Zach Lofton     7.0
Zach Norvell Jr.        15.0
Zach Randolph   32.0
Zach Smith      1.0
Zarko Cabarkapa 10.0
Zaza Pachulia   19.0
Zeljko Rebraca  13.0
Zendon Hamilton 9.0
Zhaire Smith    9.0
Zhou Qi 5.0
Zion Williamson 27.0
Zoran Dragic    4.0
Zoran Planinic  9.0
Zydrunas Ilgauskas      26.0
Zylan Cheatham  7.0
```

Highlighted in yellow are some players that have been stars of their teams. Yao Ming - all famous Chinese player who played for Houston Rockets, Zach LaVine - winner of slam dunk competition and current star at Chicago Bulls, Zion Williamson - only completed his first season in 2019 and is the next biggest basketball player - averaged 27 minutes per game in his first year (only 18 years old!)

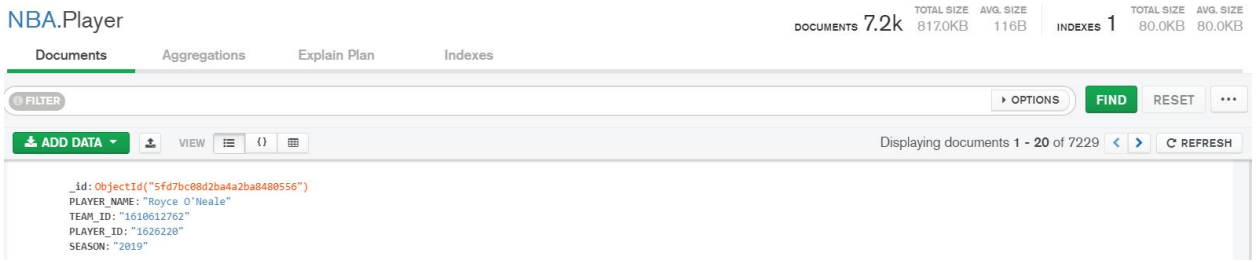
d) FG_Pct per team

Below I have calculated the average FG_PCT in a similar format for every team. Some of the top teams include Denver Nuggets, Golden State Warriors, and Oklahoma City Thunder.

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -head /FGPCTPerTeam/part-r-00000
2020-12-15 22:13:27,784 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
ATL     0.40802735
BKN     0.4180491
BOS     0.41572657
CHA     0.4068499
CHI     0.4071386
CLE     0.4050014
DAL     0.41942006
DEN     0.42738783
DET     0.40888053
GSW     0.42987737
HOU     0.41385695
IND     0.41155028
LAC     0.41819814
LAL     0.41750613
MEM     0.41421255
MIA     0.4189766
MIL     0.41287082
MIN     0.40649378
NJN     0.39107016
NOH     0.40835658
NOK     0.41900226
NOP     0.4212448
NYK     0.41144404
OKC     0.42539936
ORL     0.41887477
PHI     0.41396382
PHX     0.42761922
POR     0.4108793
SAC     0.41678816
SAS     0.4234197
SEA     0.41057312
TOR     0.41622037
UTA     0.41864195
WAS     0.41088006
```

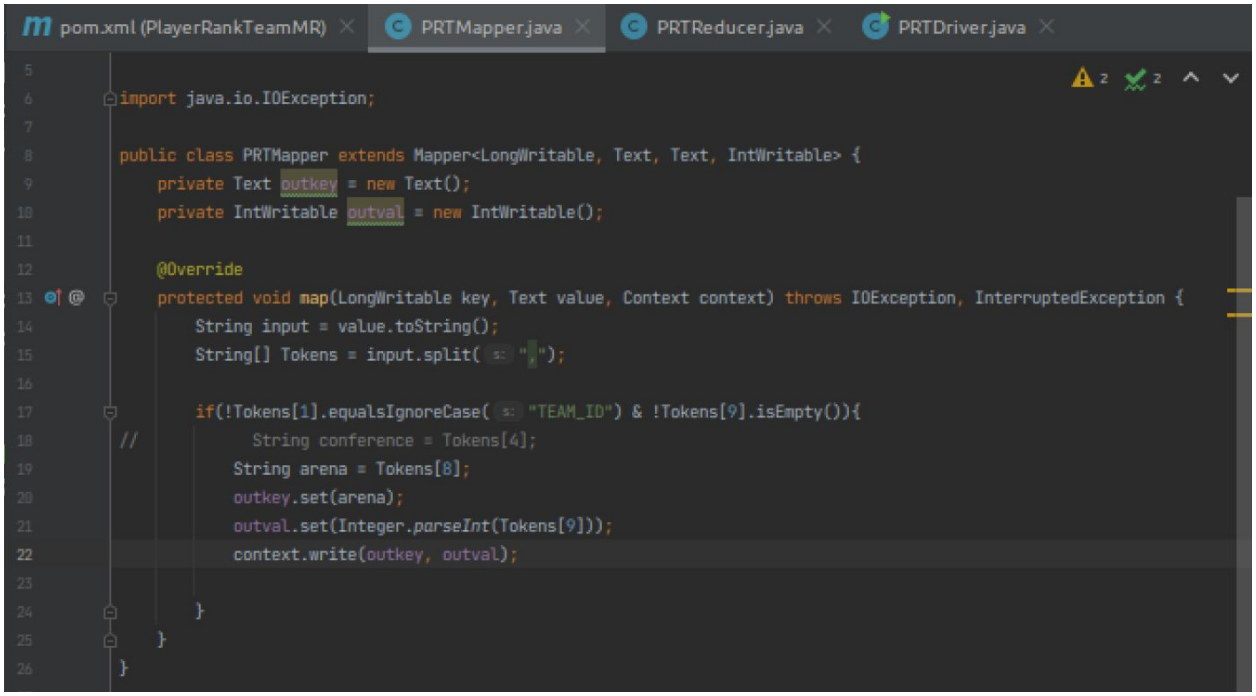
3. Players.csv

Below is a snippet of a document/row in this file

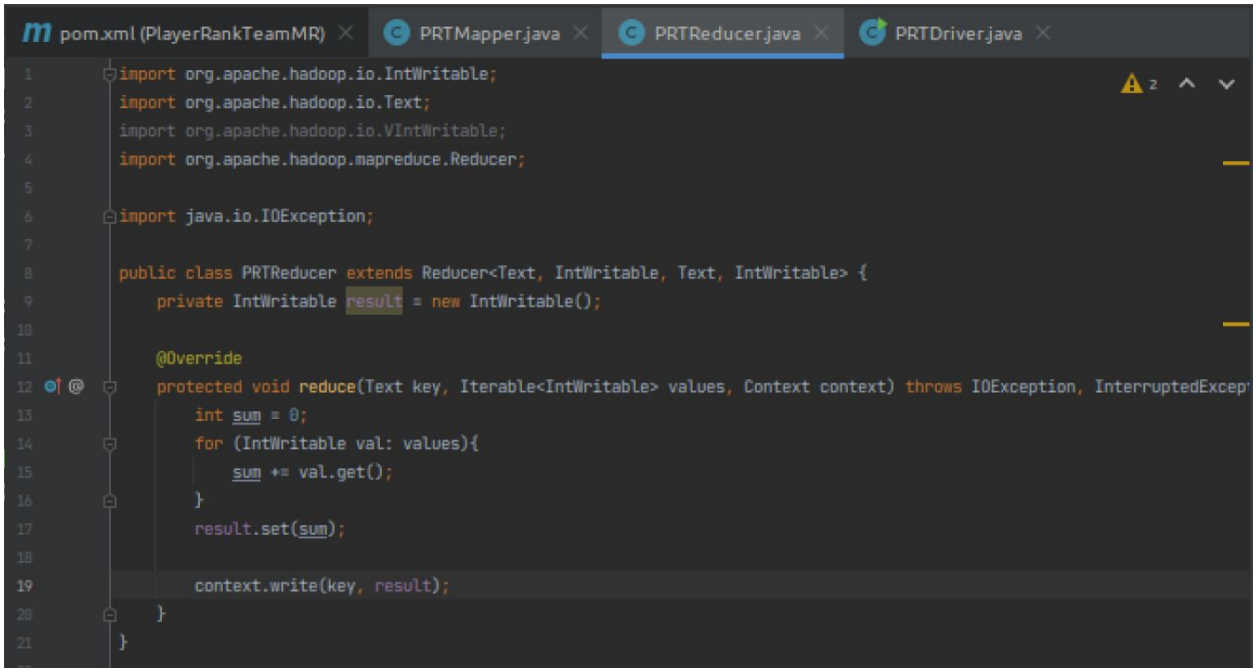


Code Snippet:

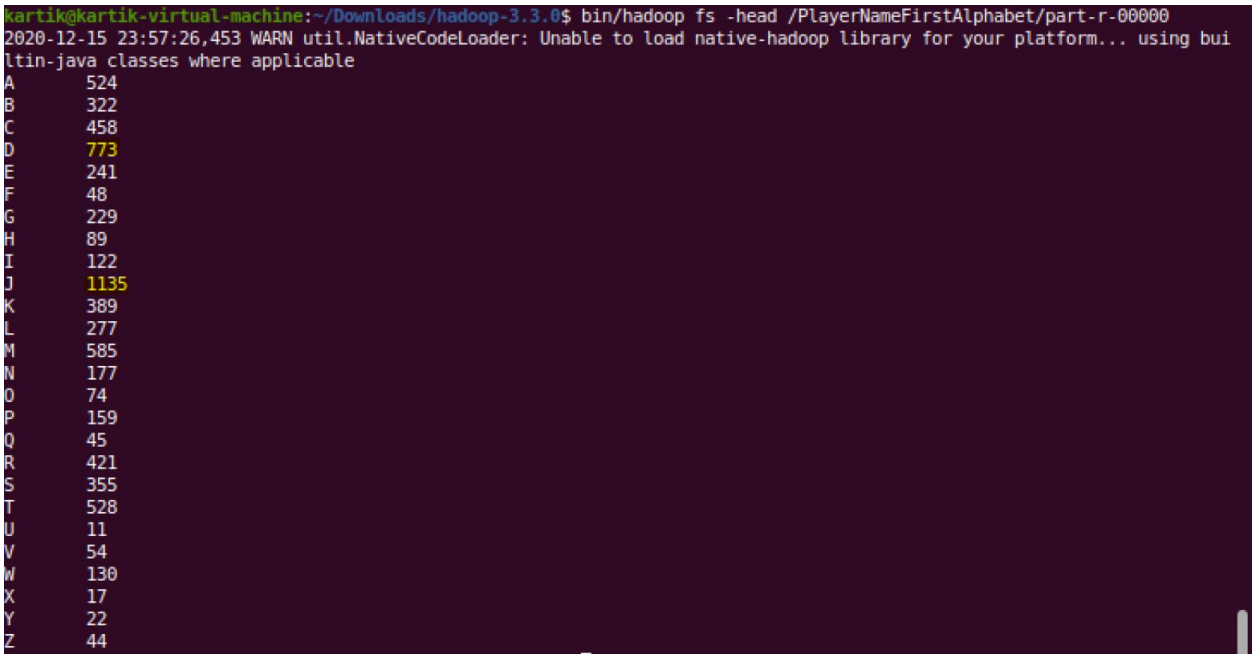
- Mapper - Same mapper has been used for all basic analysis on this file



- Reducer - Same reducer has been used for all basic analysis on this file



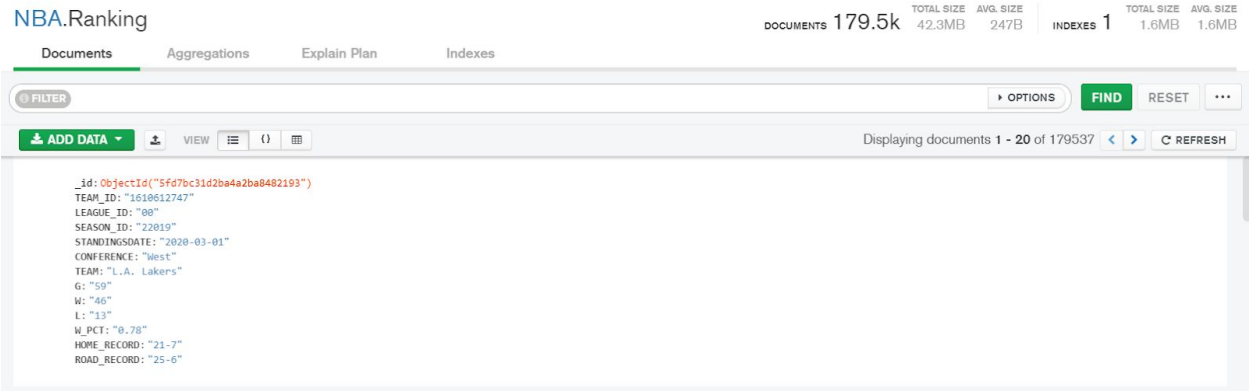
Player Names Mostly start with which alphabet?



As shown above, most names of NBA players start with J and D. A simple mapreduce implementing count with reducer as combiner yielded the above result.

4. Ranking.csv

Below is a snippet of a document/row in this file:



Games Per conference

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -head /ConferenceGames/part-r-00000
2020-12-16 00:26:30,605 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
East    5123007
West    5108108
```

East has more games! Let's settle the debate! East is clearly more competitive.

5. Teams.csv

Arena Capacities

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -head /ArenaCapacity/part-r-00000
2020-12-16 00:35:05,752 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
AT&T Center     18694
American Airlines Center        19200
AmericanAirlines Arena  19600
Amway Center    0
Bankers Life Fieldhouse 18345
Capital One Arena       20647
Chase Center    19596
Chesapeake Energy Arena 19163
FedExForum      18119
Fiserv Forum    17500
Golden 1 Center 17500
Little Caesars Arena    21000
Madison Square Garden   19763
Moda Center     19980
Pepsi Center    19099
Quicken Loans Arena     20562
Scotiabank Arena        19800
Spectrum Center 19026
Staples Center  38120
State Farm Arena        18729
TD Garden       18624
Target Center   19356
Toyota Center   18104
United Center   21711
Vivint Smart Home Arena 20148
```

Staples has the highest capacity and would have experience the biggest revenue loss due to COVID-19. Since all games were happening without crowds.

2. Intermediate Analysis

All intermediate analysis have been done on the Game_Details.csv file & Games.csv, which are the largest file and contains player statistics and season data.

a) Secondary Sort:

Sort records by player name, if the name is the same, sort according to the minutes played in that game.

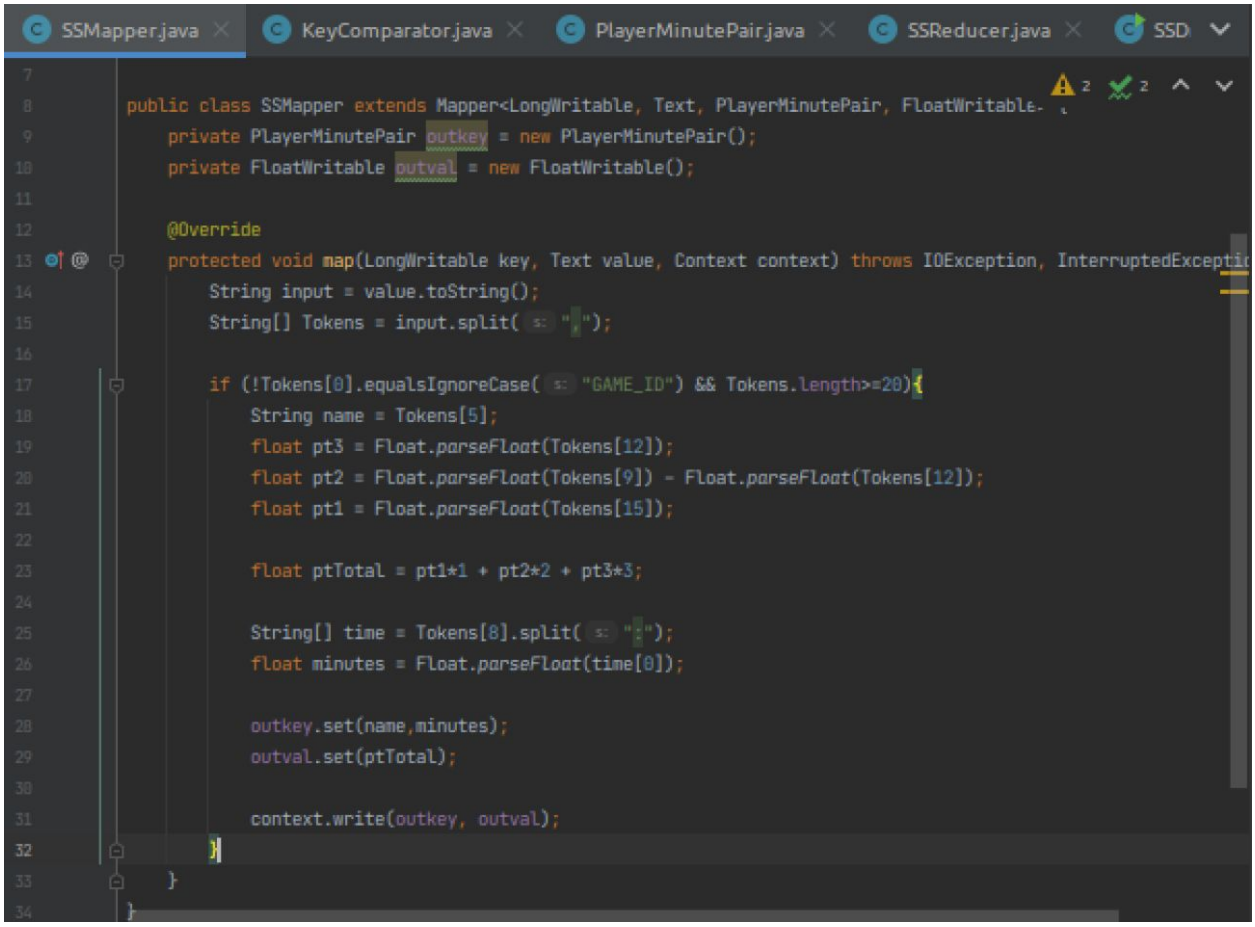
Code snippets:

- Custom Writable

PlayerMinutePair.java | SSReducer.java | SSDriver.java | NaturalKeyComparator.java

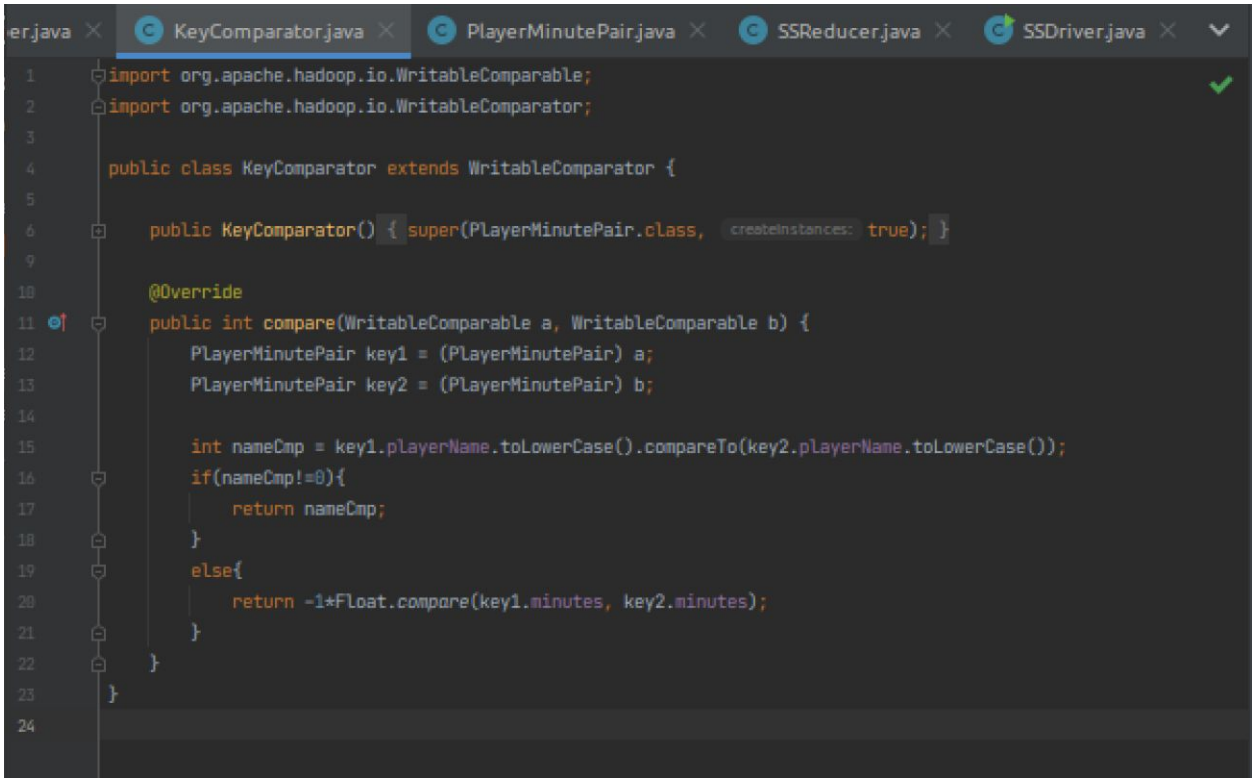
```java
import org.apache.hadoop.io.WritableComparable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.Locale;

public class PlayerMinutePair implements WritableComparable<PlayerMinutePair> {
    public String playerName;
    public float minutes;

    public PlayerMinutePair(){}

    public PlayerMinutePair(String name, float min){
        super();
        this.set(name, min);
    }

    public void set(String name, float min){
        this.playerName = (name==null)?"" : name;
        this.minutes = min;
    }

    @Override
    public void write(DataOutput dataOutput) throws IOException {
        dataOutput.writeUTF(playerName);
        dataOutput.writeFloat(minutes);
    }
```

- Mapper

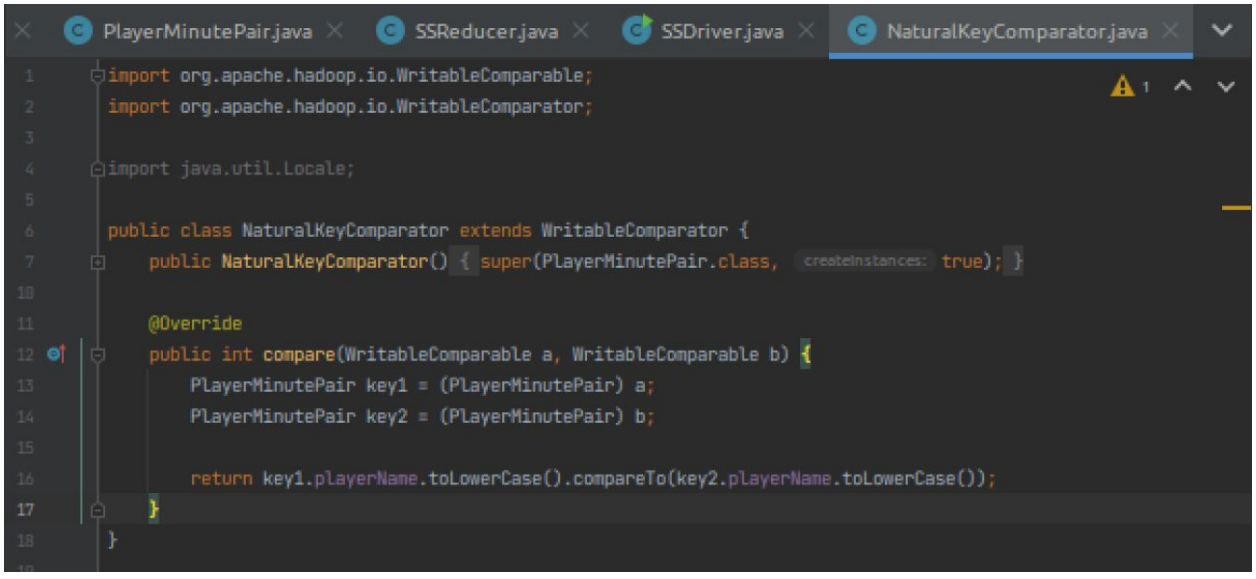


- NaturalKeyPartitioner

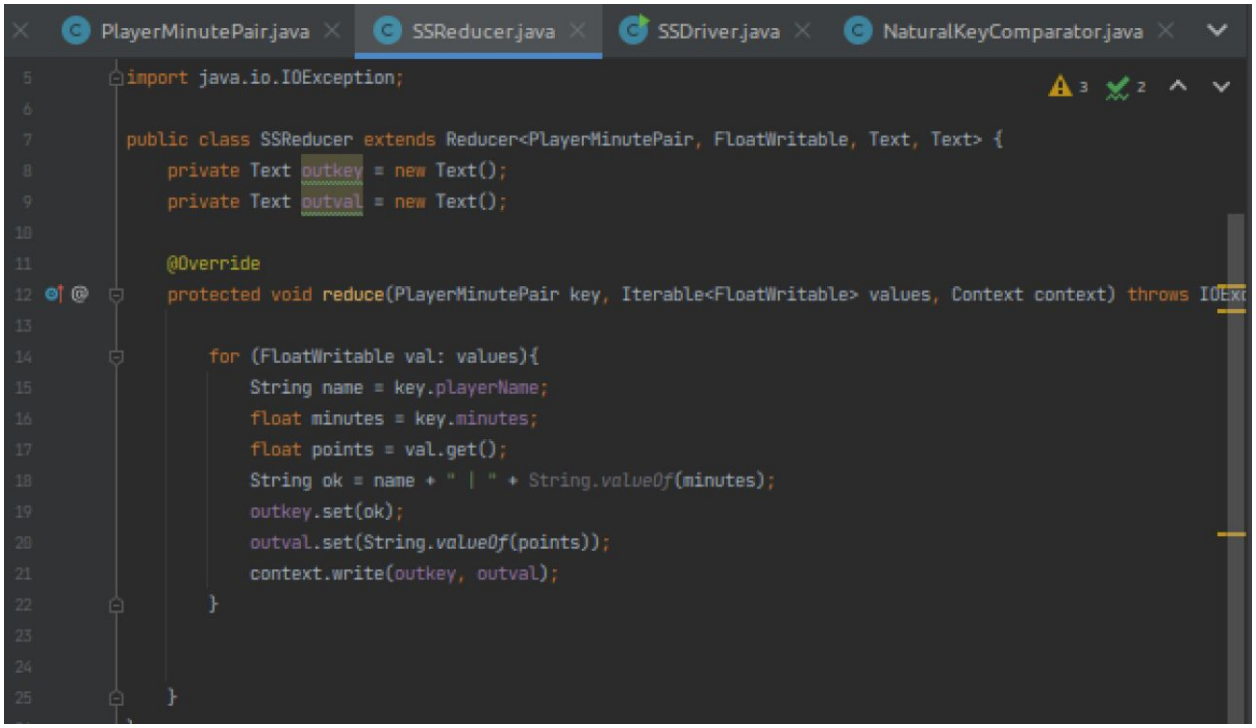
- SortComparator



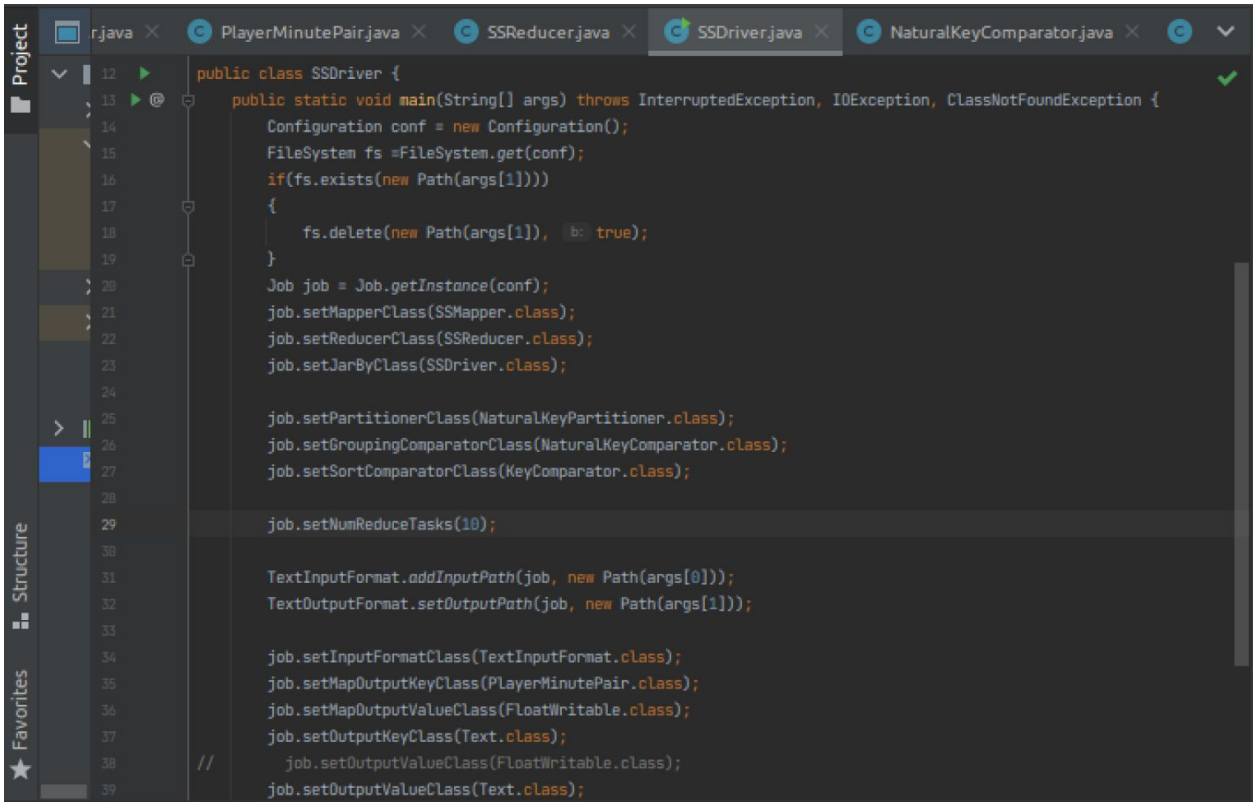
- Grouping Comparator



- Reducer



- Driver



Output

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -cat /SecSortMR/part-r-00000 | tail -5000
2020-12-16 18:24:06,273 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bu
ltin-java classes where applicable
Yao Ming | 19.0 9.0
Yao Ming | 18.0 7.0
Yao Ming | 18.0 11.0
Yao Ming | 17.0 13.0
Yao Ming | 17.0 8.0
Yao Ming | 16.0 13.0
Yao Ming | 14.0 13.0
Yao Ming | 13.0 5.0
Yao Ming | 12.0 10.0
Yao Ming | 12.0 3.0
Yao Ming | 11.0 14.0
Yao Ming | 8.0  5.0
Yao Ming | 6.0  0.0
Yao Ming | 5.0  12.0
Yao Ming | 5.0  2.0
```

```
Zach LaVine | 55.0      47.0
Zach LaVine | 49.0      41.0
Zach LaVine | 47.0      18.0
Zach LaVine | 45.0      24.0
Zach LaVine | 45.0      27.0
Zach LaVine | 44.0      23.0
Zach LaVine | 44.0      37.0
Zach LaVine | 44.0      19.0
Zach LaVine | 44.0      28.0
Zach LaVine | 44.0      20.0
Zach LaVine | 44.0      24.0
Zach LaVine | 44.0      25.0
Zach LaVine | 43.0      17.0
Zach LaVine | 43.0      11.0
Zach LaVine | 43.0      24.0
Zach LaVine | 43.0      24.0
Zach LaVine | 43.0      21.0
Zach LaVine | 42.0      17.0
Zach LaVine | 42.0      25.0
Zach LaVine | 42.0      22.0
Zach LaVine | 42.0      11.0
Zach LaVine | 42.0      18.0
Zach LaVine | 41.0      26.0
Zach LaVine | 41.0      9.0
Zach LaVine | 41.0      23.0
Zach LaVine | 41.0      10.0
Zach LaVine | 40.0      26.0
Zach LaVine | 40.0      16.0
```

```
Zach Randolph | 55.0    34.0
Zach Randolph | 50.0    16.0
Zach Randolph | 49.0    34.0
Zach Randolph | 48.0    21.0
Zach Randolph | 47.0    34.0
Zach Randolph | 47.0    27.0
Zach Randolph | 47.0    38.0
Zach Randolph | 47.0    28.0
Zach Randolph | 47.0    25.0
Zach Randolph | 47.0    32.0
Zach Randolph | 46.0    21.0
Zach Randolph | 46.0    27.0
Zach Randolph | 46.0    31.0
Zach Randolph | 46.0    28.0
Zach Randolph | 46.0    42.0
Zach Randolph | 46.0    24.0
Zach Randolph | 46.0    31.0
Zach Randolph | 46.0    40.0
Zach Randolph | 45.0    14.0
Zach Randolph | 45.0    15.0
Zach Randolph | 45.0    25.0
Zach Randolph | 45.0    24.0
Zach Randolph | 45.0    24.0
Zach Randolph | 45.0    22.0
Zach Randolph | 45.0    33.0
Zach Randolph | 45.0    29.0
Zach Randolph | 44.0    17.0
```

```
Zion Williamson | 33.0  28.0
Zion Williamson | 33.0  35.0
Zion Williamson | 32.0  29.0
Zion Williamson | 32.0  21.0
Zion Williamson | 31.0  20.0
Zion Williamson | 31.0  32.0
Zion Williamson | 30.0  22.0
Zion Williamson | 30.0  24.0
Zion Williamson | 29.0  25.0
Zion Williamson | 29.0  14.0
Zion Williamson | 28.0  24.0
Zion Williamson | 27.0  21.0
Zion Williamson | 27.0  16.0
Zion Williamson | 27.0  31.0
Zion Williamson | 26.0  29.0
Zion Williamson | 25.0  21.0
Zion Williamson | 23.0  26.0
Zion Williamson | 20.0  15.0
Zion Williamson | 18.0  22.0
Zoran Dragic | 40.0     22.0
Zoran Dragic | 8.0      0.0
Zoran Dragic | 4.0      0.0
Zoran Dragic | 4.0      0.0
```

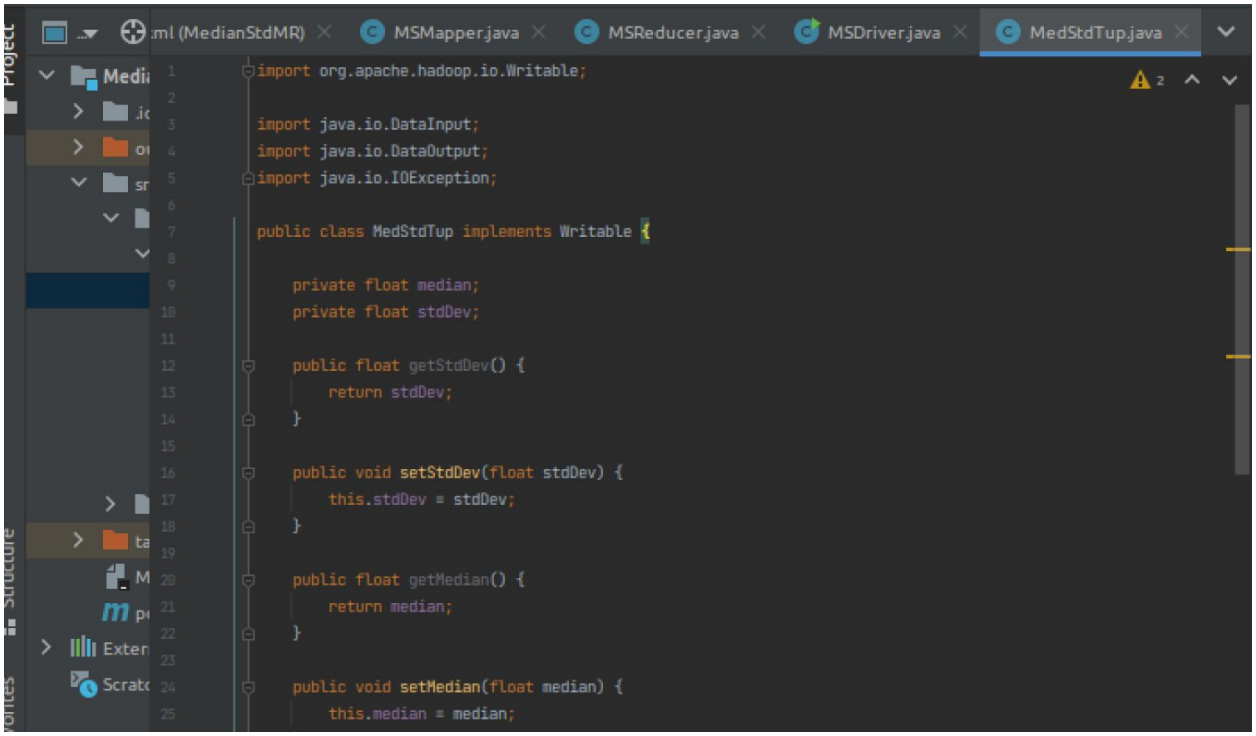
As seen in all outputs above, the result is first sorted as per the player name. If the name is same, the records are sorted by minutes played - a float value shown after “|”. The right most value is the total points scored in that game by that player.

b) Median & Std (22 sec)

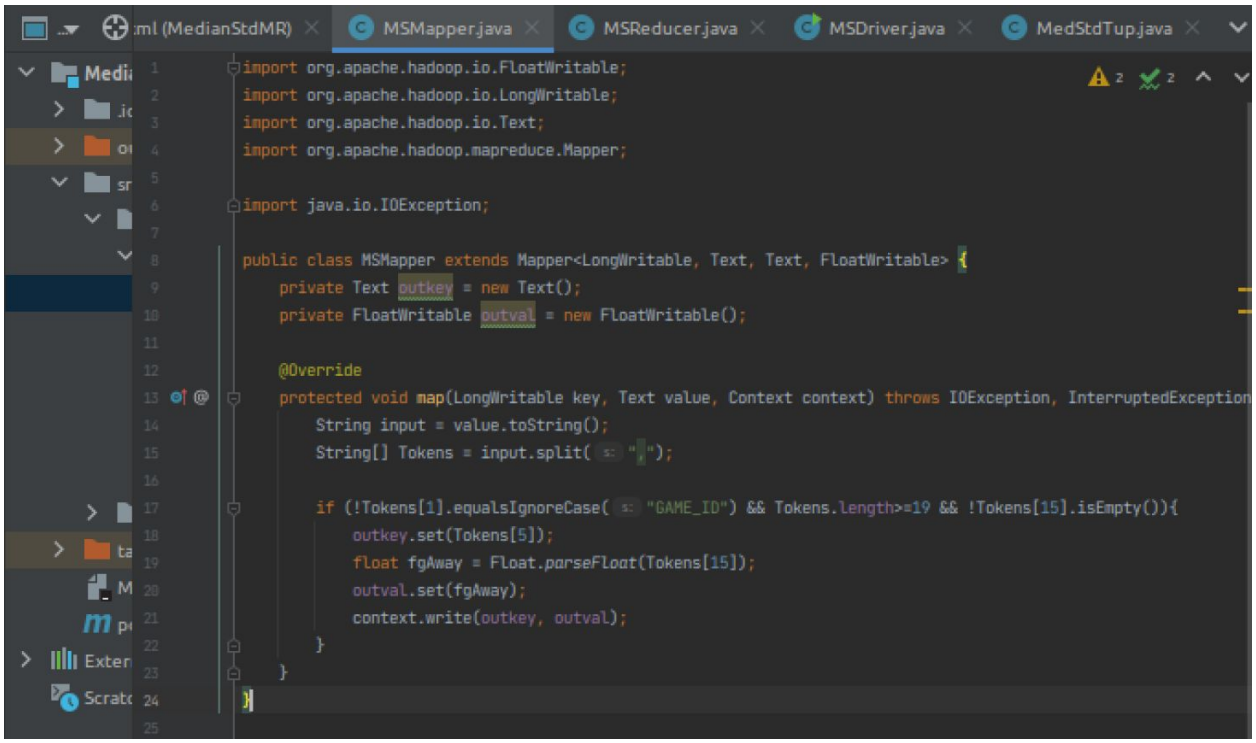
Calculate the average Field Goal Percentage for away teams, per season.

Code Snippets:

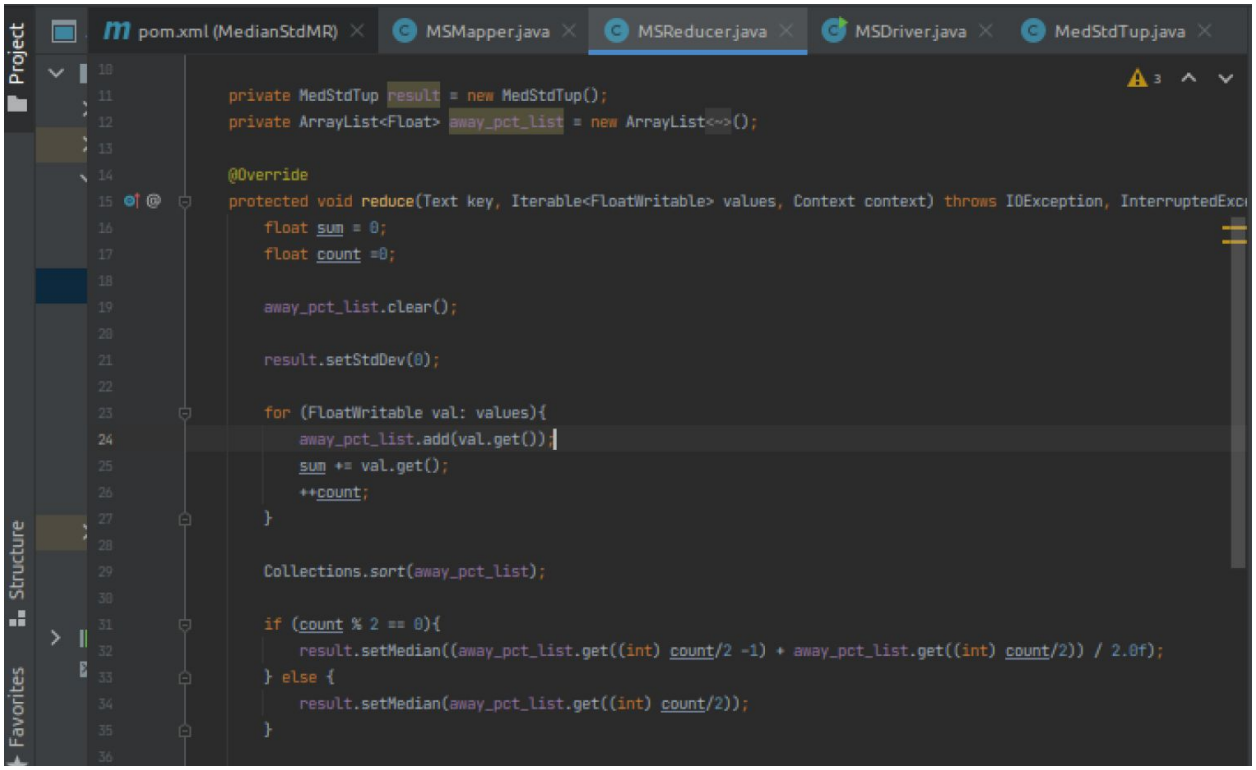
- Custom Writable



- Mapper

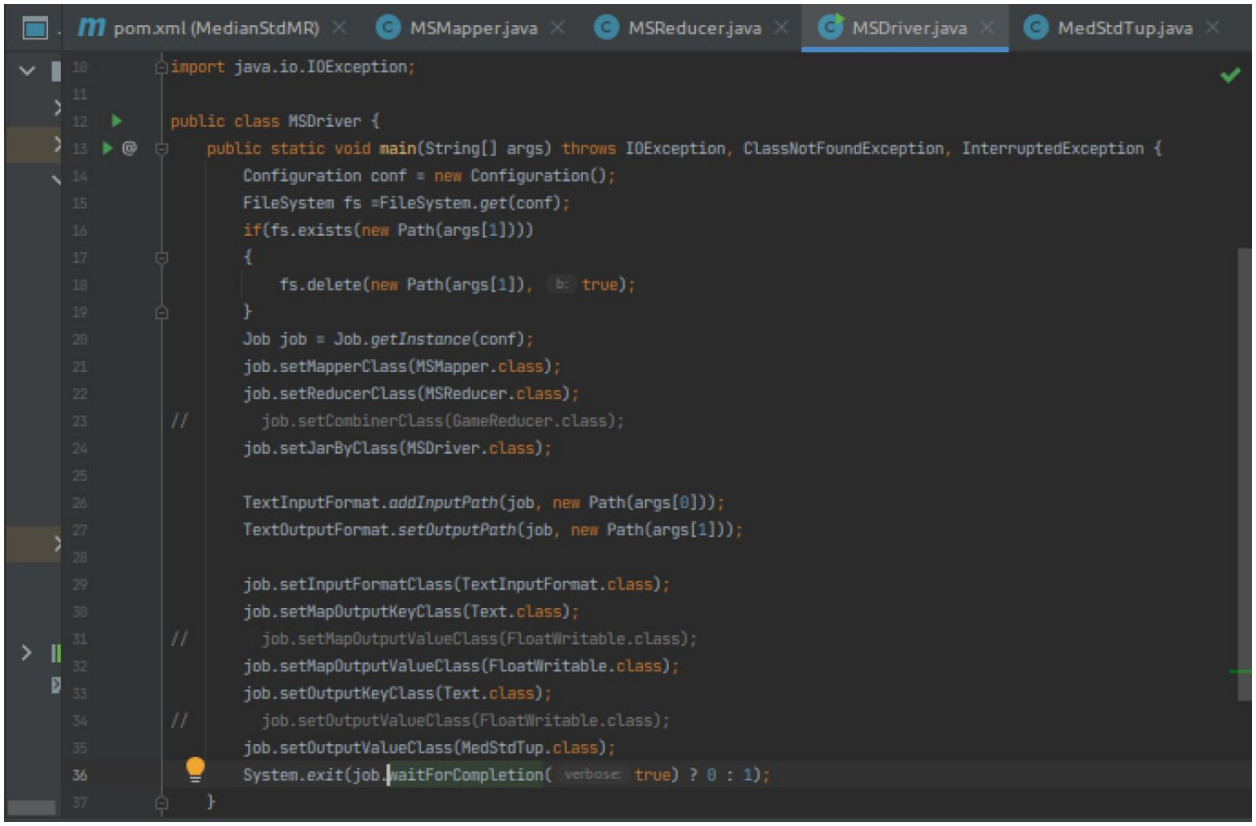


- Reducer

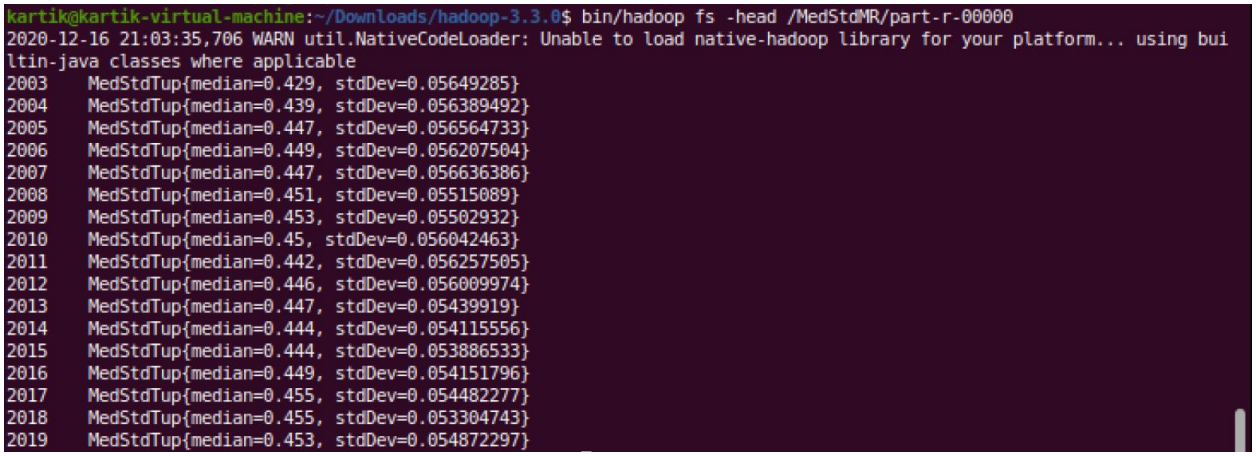

pom.xml (MedianStdMR)
MSMapper.java
MSReducer.java
MSDriver.java
MedStdTup.java
Project
Structure
Favorites
private MedStdTup result = new MedStdTup();
private ArrayList<Float> away_pct_list = new ArrayList<>();
@Override
protected void reduce(Text key, Iterable<FloatWritable> values, Context context) throws IOException, InterruptedExc
float sum = 0;
float count =0;
away_pct_list.clear();
result.setStdDev(0);
for (FloatWritable val: values){
away_pct_list.add(val.get());
sum += val.get();
++count;
}
Collections.sort(away_pct_list);
if (count % 2 == 0){
result.setMedian((away_pct_list.get((int) count/2 -1) + away_pct_list.get((int) count/2)) / 2.0f);
} else {
result.setMedian(away_pct_list.get((int) count/2));
}


pom.xml (MedianStdMR)
MSMapper.java
MSReducer.java
MSDriver.java
MedStdTup.java
Collections.sort(away_pct_list);
if (count % 2 == 0){
result.setMedian((away_pct_list.get((int) count/2 -1) + away_pct_list.get((int) count/2)) / 2.0f);
} else {
result.setMedian(away_pct_list.get((int) count/2));
}
float mean = sum / count;
float sumOfSquares = 0.0f;
for (Float f: away_pct_list){
sumOfSquares += (f - mean)* (f - mean);
}
result.setStdDev((float) Math.sqrt(sumOfSquares / (count-1)));
context.write(key, result);
}
}

- Driver



Output:



The Median has increased over the year. And the Standard Deviation has gone down. The league is surely becoming more competitive by the year!

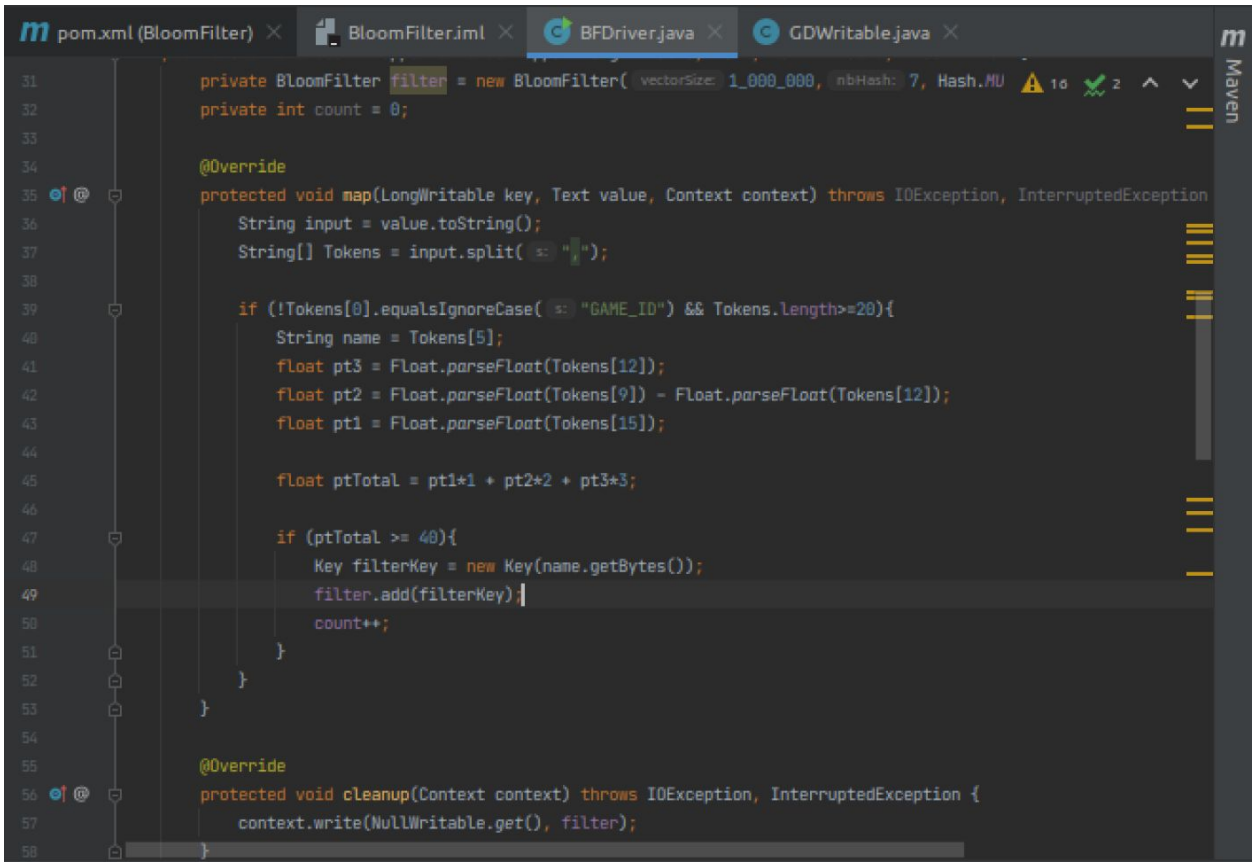
c) Bloom Filter (If player scored >= 40 points in game , show the player names)

Code Snippets:

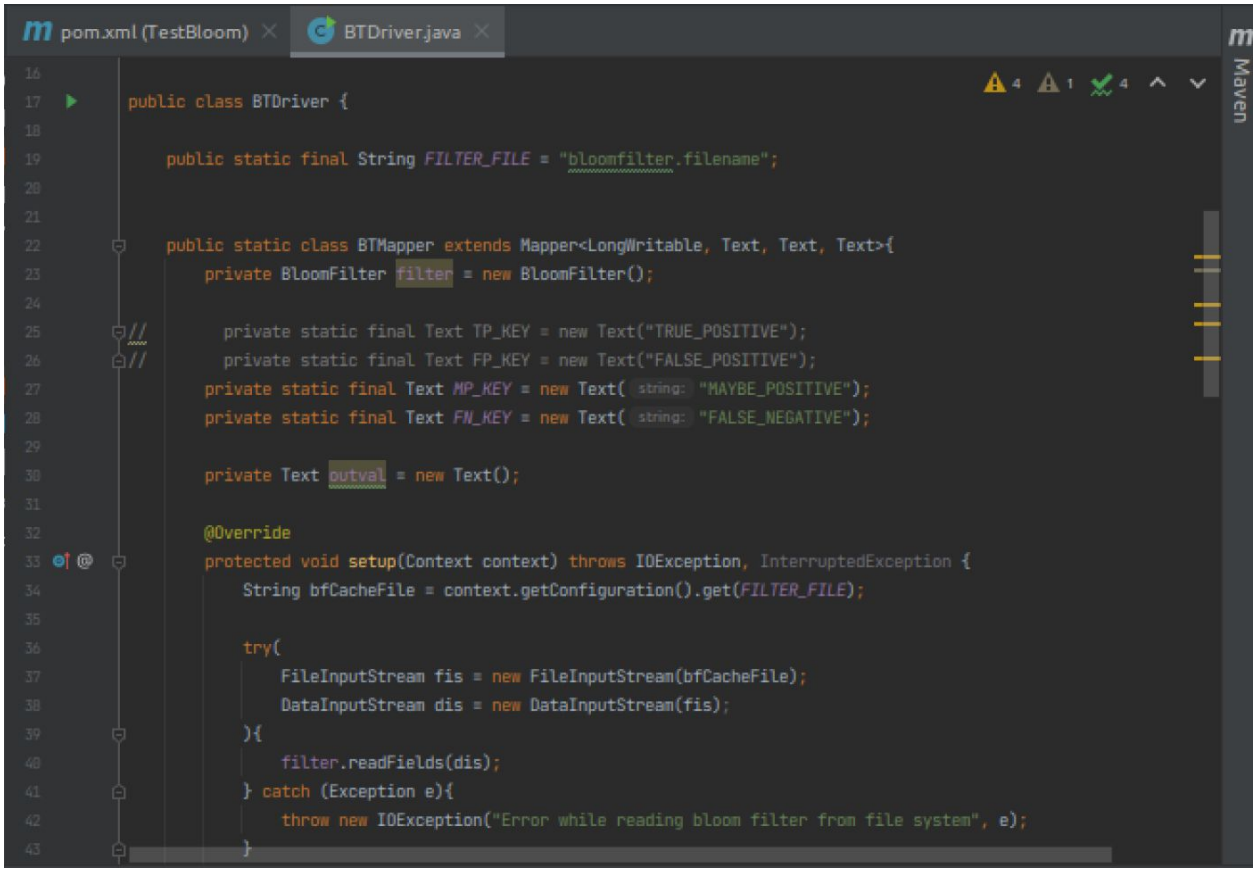
- Filter training

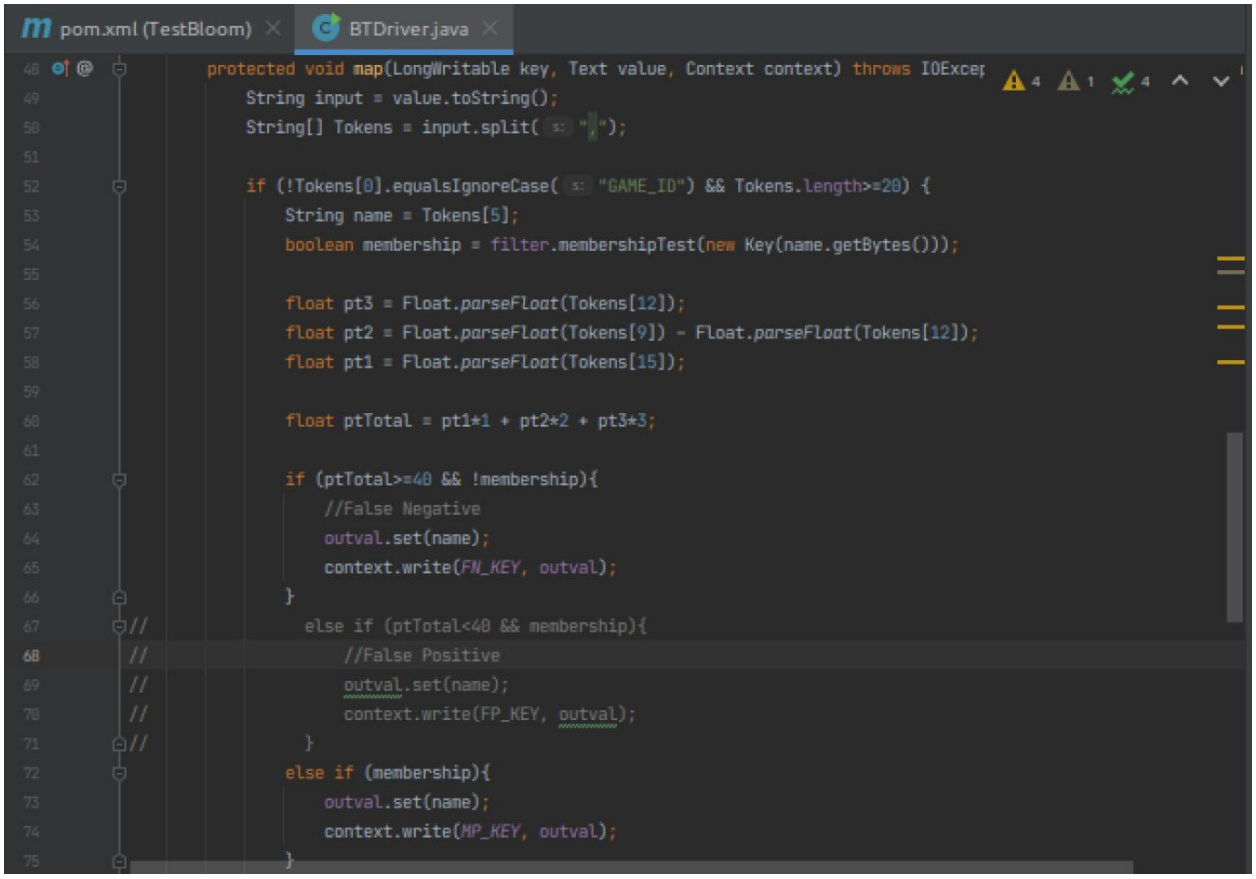
pom.xml (BloomFilter) | BloomFilter.iml | BFDriver.java | GDWritable.java

```java
import org.apache.hadoop.io.Writable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class GDWritable implements Writable {
    private String gameId;
    private String playerName;
    private float totalPts;

    @Override
    public void write(DataOutput dataOutput) throws IOException {
        dataOutput.writeUTF(gameId);
        dataOutput.writeUTF(playerName);
        dataOutput.writeFloat(totalPts);
    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
        gameId = dataInput.readUTF();
        playerName = dataInput.readUTF();
        totalPts = dataInput.readFloat();
    }
```



- Driver with Mapper - Run the Bloom Filter Test with stored file


pom.xml (TestBloom)
BTDriver.java
Maven
public class BTDriver {
public static final String FILTER_FILE = "bloomfilter.filename";
public static class BTMapper extends Mapper<LongWritable, Text, Text, Text>{
private BloomFilter filter = new BloomFilter();
// private static final Text TP_KEY = new Text("TRUE_POSITIVE");
// private static final Text FP_KEY = new Text("FALSE_POSITIVE");
private static final Text MP_KEY = new Text(string: "MAYBE_POSITIVE");
private static final Text FN_KEY = new Text(string: "FALSE_NEGATIVE");
private Text outval = new Text();
@Override
protected void setup(Context context) throws IOException, InterruptedException {
String bfCacheFile = context.getConfiguration().get(FILTER_FILE);
try(
FileInputStream fis = new FileInputStream(bfCacheFile);
DataInputStream dis = new DataInputStream(fis);
){
filter.readFields(dis);
} catch (Exception e){
throw new IOException("Error while reading bloom filter from file system", e);
}


pom.xml (TestBloom)
BTDriver.java
protected void map(LongWritable key, Text value, Context context) throws IOExcep
String input = value.toString();
String[] Tokens = input.split(s: ",");
if (!Tokens[0].equalsIgnoreCase(s: "GAME_ID") && Tokens.length>=20) {
String name = Tokens[5];
boolean membership = filter.membershipTest(new Key(name.getBytes()));
float pt3 = Float.parseFloat(Tokens[12]);
float pt2 = Float.parseFloat(Tokens[9]) - Float.parseFloat(Tokens[12]);
float pt1 = Float.parseFloat(Tokens[15]);
float ptTotal = pt1*1 + pt2*2 + pt3*3;
if (ptTotal>=40 && !membership){
//False Negative
outval.set(name);
context.write(FN_KEY, outval);
}
// else if (ptTotal<40 && membership){
// //False Positive
// outval.set(name);
// context.write(FP_KEY, outval);
// }
else if (membership){
outval.set(name);
context.write(MP_KEY, outval);
}

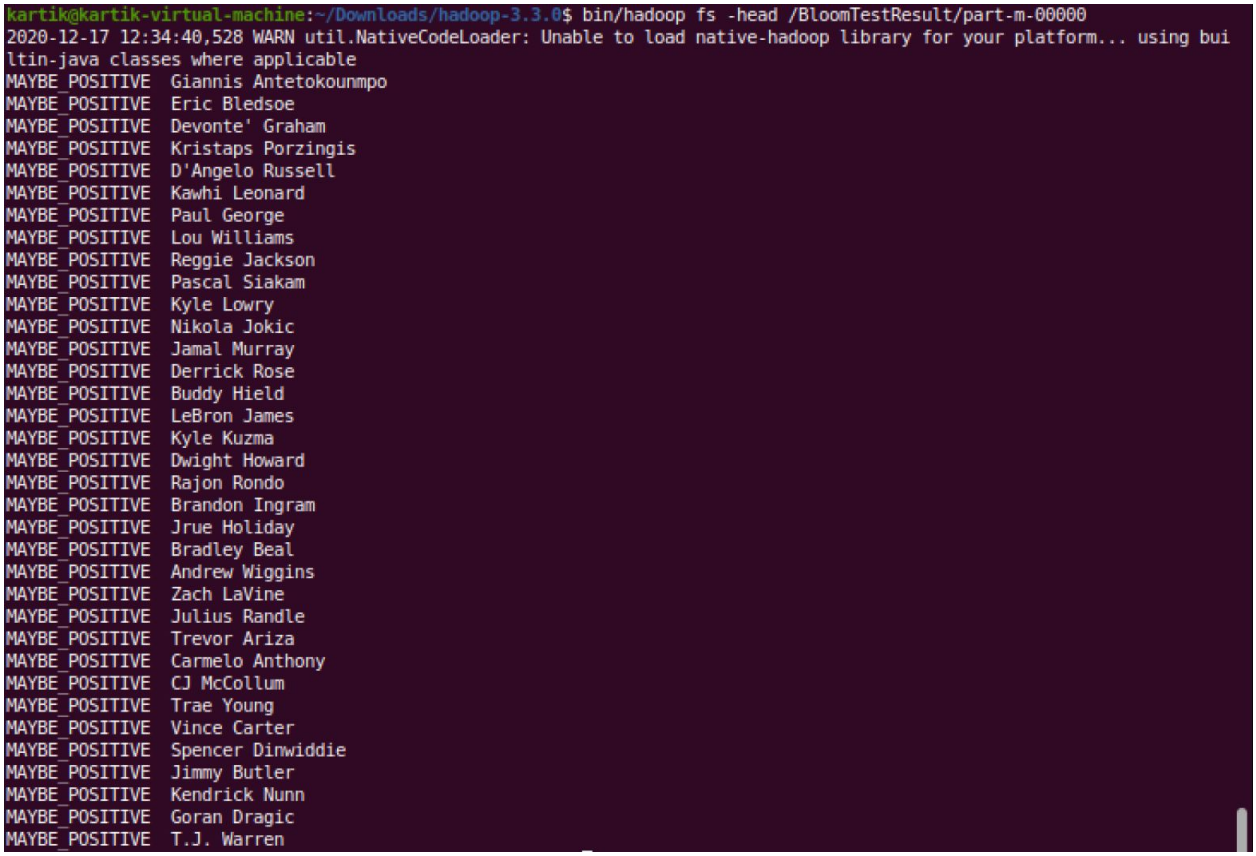
pom.xml (TestBloom) | BTDriver.java

```
81  public static void main(String[] args) throws IOException, ClassNotFoundException, I
82      Configuration conf = new Configuration();
83      Job job = Job.getInstance(conf, jobName: "Test Bloom");
84      job.setJarByClass(BTDriver.class);
85
86      Path datasetPath = new Path(args[0]);
87      Path bfPath = new Path(args[1]);
88      Path outputPath = new Path(args[2]);
89
90      job.addCacheFile(bfPath.toUri());
91      job.getConfiguration().set(FILTER_FILE, bfPath.getName());
92
93      job.setMapperClass(BTMapper.class);
94      job.setInputFormatClass(TextInputFormat.class);
95      job.setMapOutputKeyClass(Text.class);
96      job.setMapOutputValueClass(Text.class);
97
98      job.setNumReduceTasks(0);
99
100     FileInputFormat.setInputPaths(job, datasetPath);
101     FileOutputFormat.setOutputPath(job, outputPath);
102
103     System.exit(job.waitForCompletion( verbose: true) ? 0: 1);
104 }
105 }
106
```

Output:
Filter stored like:

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -ls /Bloom
2020-12-17 11:58:52,923 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
Found 1 items
-rw-r--r--   1 kartik supergroup     125013 2020-12-17 11:58 /Bloom/filter
```

After running Bloom Filter Test, we get these player names.
First on the list is MVP from the last two seasons 2018-19 & 2019-20, Giannis AKA Greek freak!
Everyone of these players is a famous player. They have carried the teams to wins single-handedly multiple times. My personal favorite are Rajon Rondo & Kawhi Leonard

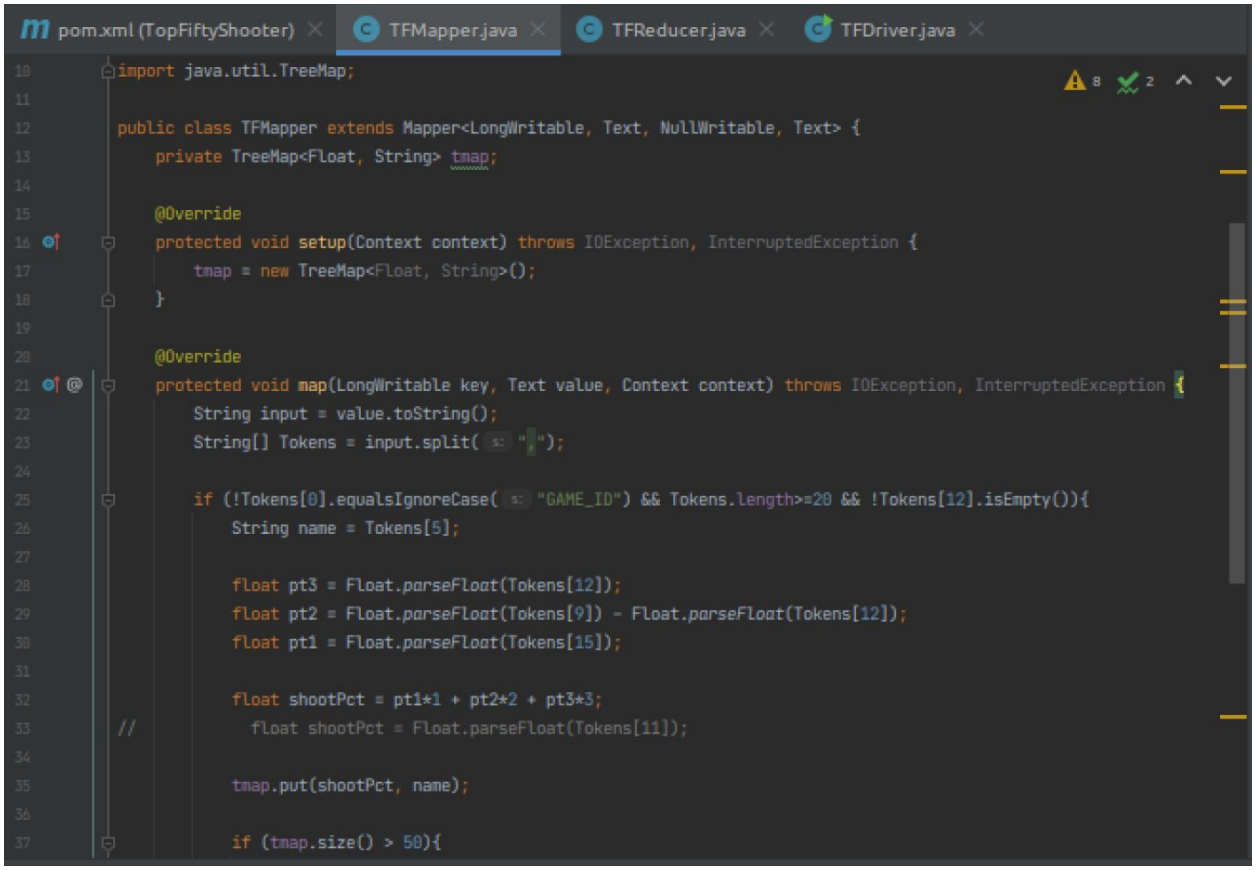


d) Top 50 Filter - Highest FG_Pct in a game

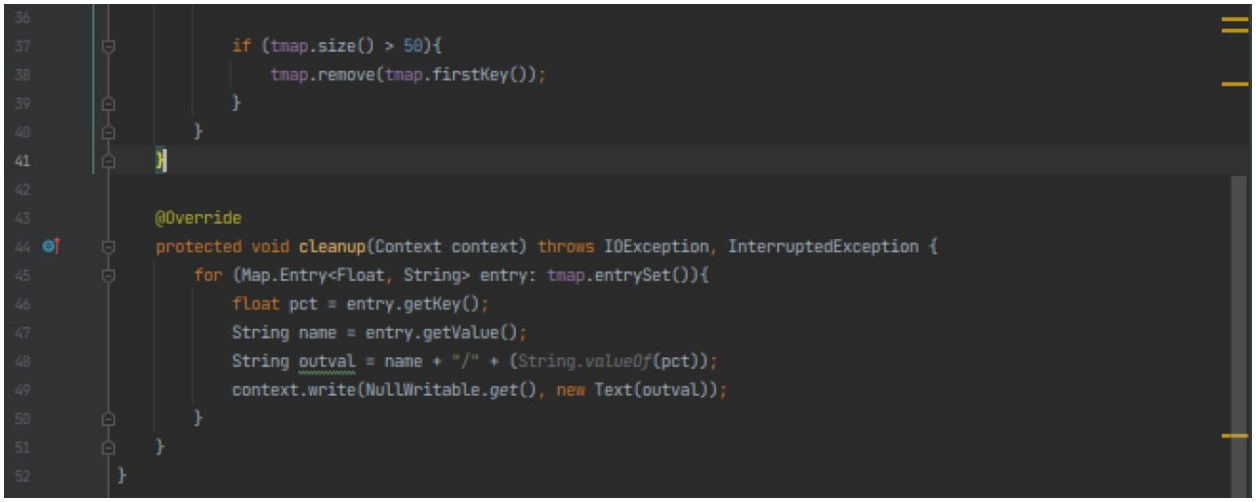
Calculate the players with Top 50 shooting percentage in a game.

Code Snippet:

- Mapper

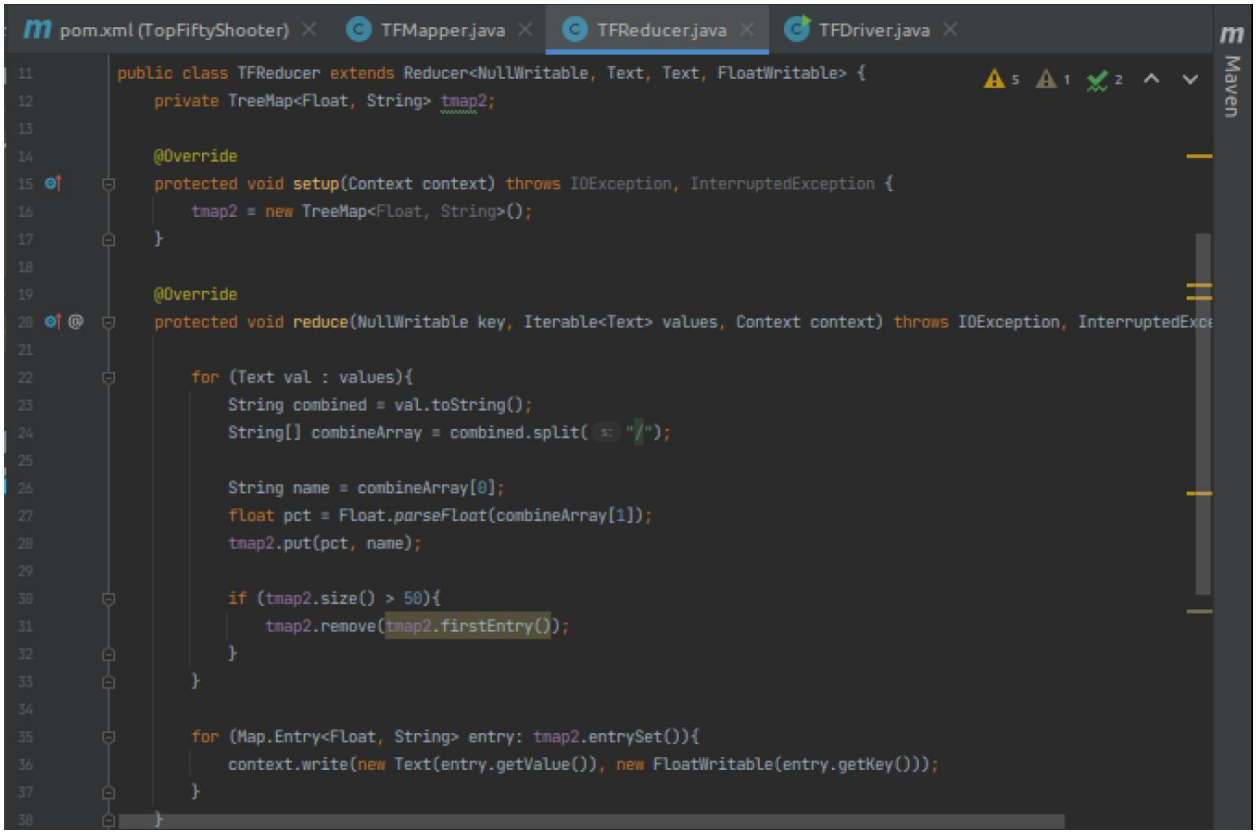


```
import java.util.TreeMap;

public class TFMapper extends Mapper<LongWritable, Text, NullWritable, Text> {
    private TreeMap<Float, String> tmap;

    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        tmap = new TreeMap<Float, String>();
    }

    @Override
    protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
        String input = value.toString();
        String[] Tokens = input.split( s: ",");

        if (!Tokens[0].equalsIgnoreCase( s: "GAME_ID") && Tokens.length>=20 && !Tokens[12].isEmpty()){
            String name = Tokens[5];

            float pt3 = Float.parseFloat(Tokens[12]);
            float pt2 = Float.parseFloat(Tokens[9]) - Float.parseFloat(Tokens[12]);
            float pt1 = Float.parseFloat(Tokens[15]);

            float shootPct = pt1*1 + pt2*2 + pt3*3;
//            float shootPct = Float.parseFloat(Tokens[11]);

            tmap.put(shootPct, name);

            if (tmap.size() > 50){
```

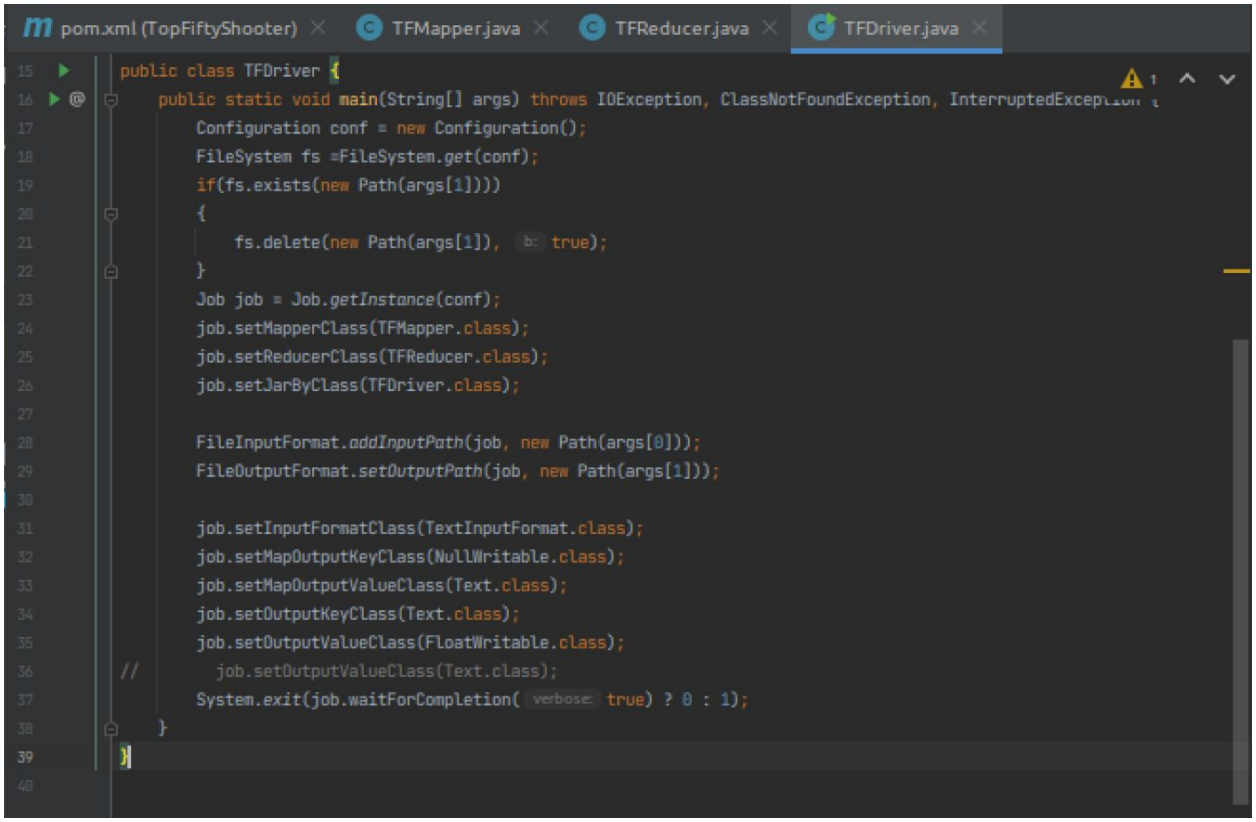


```
                tmap.remove(tmap.firstKey());
            }
        }
    }

    @Override
    protected void cleanup(Context context) throws IOException, InterruptedException {
        for (Map.Entry<Float, String> entry: tmap.entrySet()){
            float pct = entry.getKey();
            String name = entry.getValue();
            String outval = name + "/" + (String.valueOf(pct));
            context.write(NullWritable.get(), new Text(outval));
        }
    }
}
```

- Reducer


pom.xml (TopFiftyShooter)
TFMapper.java
TFReducer.java
TFDriver.java
Maven
public class TFReducer extends Reducer<NullWritable, Text, Text, FloatWritable> {
private TreeMap<Float, String> tmap2;
@Override
protected void setup(Context context) throws IOException, InterruptedException {
tmap2 = new TreeMap<Float, String>();
}
@Override
protected void reduce(NullWritable key, Iterable<Text> values, Context context) throws IOException, InterruptedExce
for (Text val : values){
String combined = val.toString();
String[] combineArray = combined.split(s: "/");
String name = combineArray[0];
float pct = Float.parseFloat(combineArray[1]);
tmap2.put(pct, name);
if (tmap2.size() > 50){
tmap2.remove(tmap2.firstEntry());
}
}
for (Map.Entry<Float, String> entry: tmap2.entrySet()){
context.write(new Text(entry.getValue()), new FloatWritable(entry.getKey()));
}
}

- Driver


pom.xml (TopFiftyShooter)
TFMapper.java
TFReducer.java
TFDriver.java
public class TFDriver {
public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {
Configuration conf = new Configuration();
FileSystem fs =FileSystem.get(conf);
if(fs.exists(new Path(args[1])))
{
fs.delete(new Path(args[1]), b: true);
}
Job job = Job.getInstance(conf);
job.setMapperClass(TFMapper.class);
job.setReducerClass(TFReducer.class);
job.setJarByClass(TFDriver.class);
FileInputFormat.addInputPath(job, new Path(args[0]));
FileOutputFormat.setOutputPath(job, new Path(args[1]));
job.setInputFormatClass(TextInputFormat.class);
job.setMapOutputKeyClass(NullWritable.class);
job.setMapOutputValueClass(Text.class);
job.setOutputKeyClass(Text.class);
job.setOutputValueClass(FloatWritable.class);
// job.setOutputValueClass(Text.class);
System.exit(job.waitForCompletion(verbose: true) ? 0 : 1);
}
}

Believe it or not, Shaquille O'Neal had the highest FG_PCT in game all these years! 93.8%
The guy with a value of 1, probably shot just once and made it!

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -head /Top50FGPCT/part-r-00000
2020-12-17 14:39:02,984 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
Andre Drummond  0.692
LeBron James    0.696
DeAndre Jordan  0.7
Al Jefferson    0.704
Eric Bledsoe    0.706
LeBron James    0.708
Andre Miller    0.71
Jerome Jordan   0.714
Nikola Vucevic  0.72
Chris Bosh      0.722
JR Smith        0.727
Paul George     0.731
Thaddeus Young  0.733
James Harden    0.737
Anthony Davis   0.739
Stephen Curry   0.741
Bismack Biyombo 0.75
James Harden    0.76
Anderson Varejao        0.762
Al Horford      0.765
Blake Griffin   0.769
Karl-Anthony Towns      0.773
Brian Roberts   0.778
Blake Griffin   0.783
Kobe Bryant     0.786
Klay Thompson   0.789
Blake Griffin   0.792
Al Horford      0.8
Samuel Dalembert        0.81
Paul Pierce     0.813
Chris Bosh      0.818
Dirk Nowitzki   0.824
LaMarcus Aldridge       0.826
Lou Williams    0.833
Amar'e Stoudemire       0.842
Serge Ibaka     0.846
Klay Thompson   0.85
```

```
Klay Thompson   0.85
Josh Powell     0.857
David Lee       0.867
Corey Brewer    0.875
Serge Ibaka     0.882
JJ Hickson      0.889
Tyson Chandler  0.9
John Henson     0.909
Andray Blatche  0.917
Tyson Chandler  0.923
LeBron James    0.929
Jonas Valanciunas       0.933
Shaquille O'Neal        0.938
Timofey Mozgov  1.0
```

Using the same code, to calculate the highest points score in a game.

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -cat /Top50FGPCT/part-r-00000 | tail -n 10
2020-12-17 15:10:06,564 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
DeMarcus Cousins         56.0
Kyrie Irving    57.0
Russell Westbrook        58.0
Anthony Davis   59.0
Kobe Bryant     60.0
LeBron James    61.0
Carmelo Anthony 62.0
Kobe Bryant     65.0
Devin Booker    70.0
Kobe Bryant     81.0
```

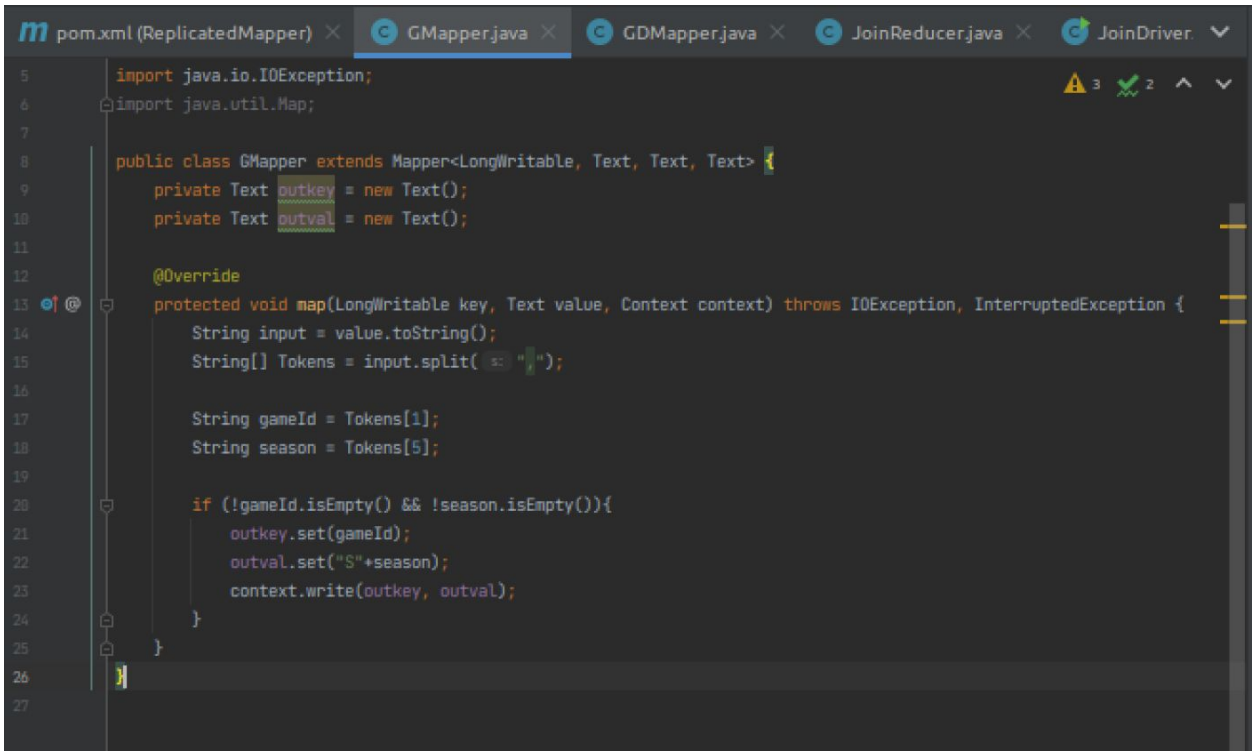
We see Kobe Bryant 3 times here - a true legend. He led the Lakers to multiple championships. Sadly, he passed away in 2020 due to a helicopter crash. His legacy remains at the heart of the NBA.

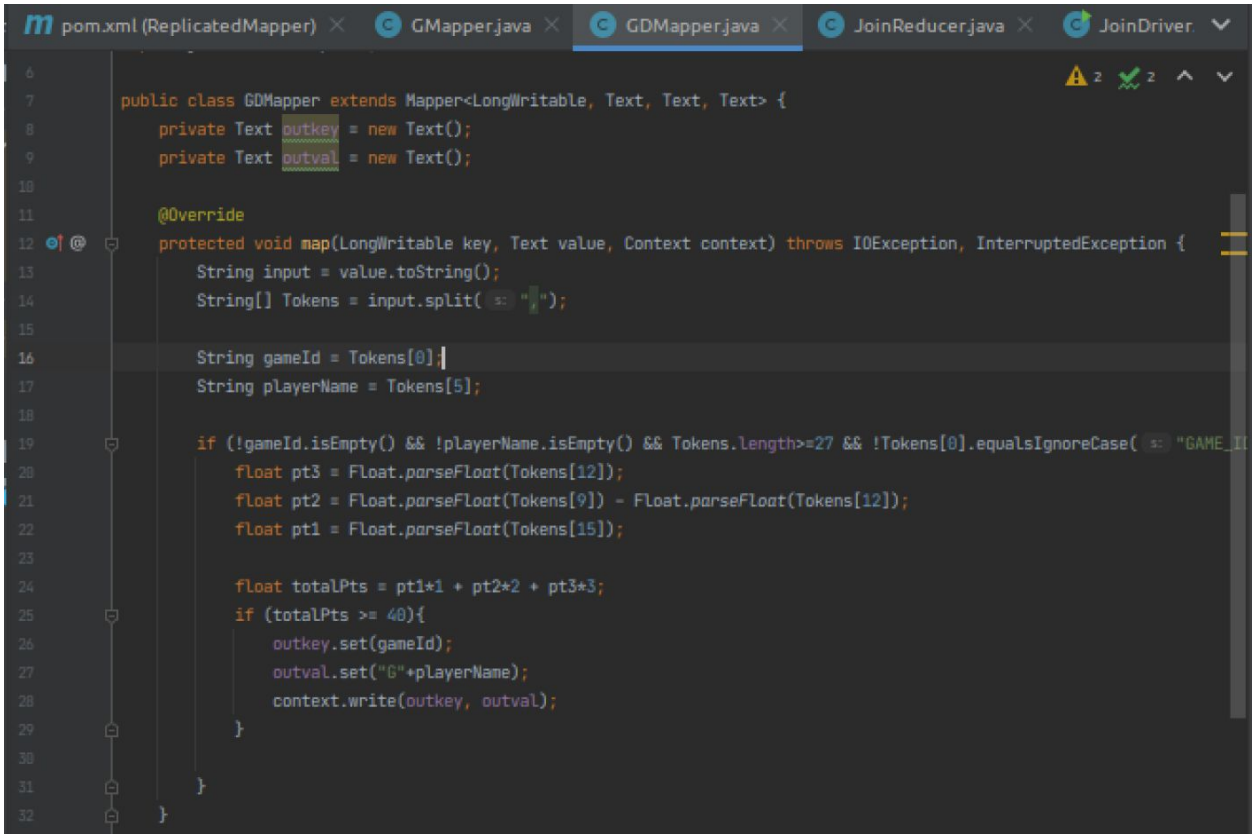
e) Joins - Reduce Side Join

Perform Inner join on games.csv & games_details.csv to get season and playername mapping.

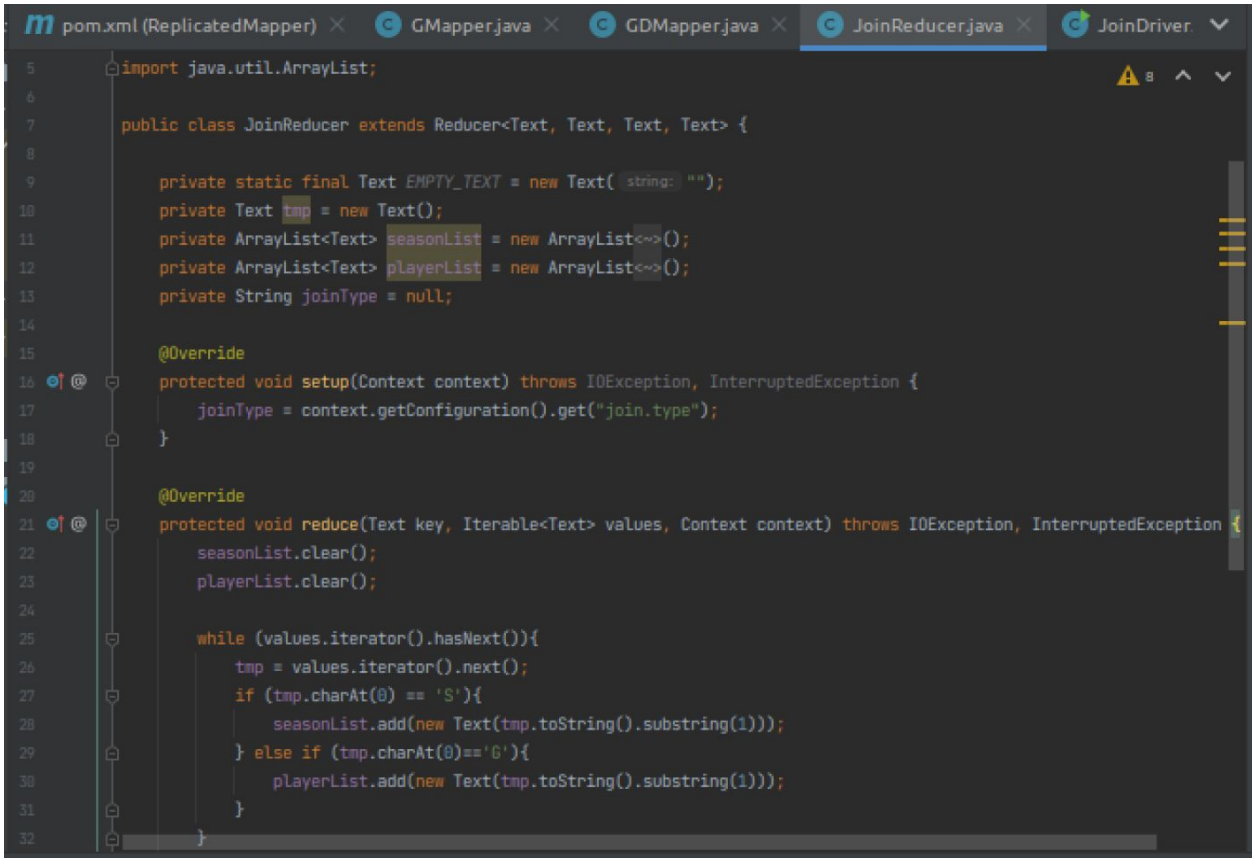
Code snippet:

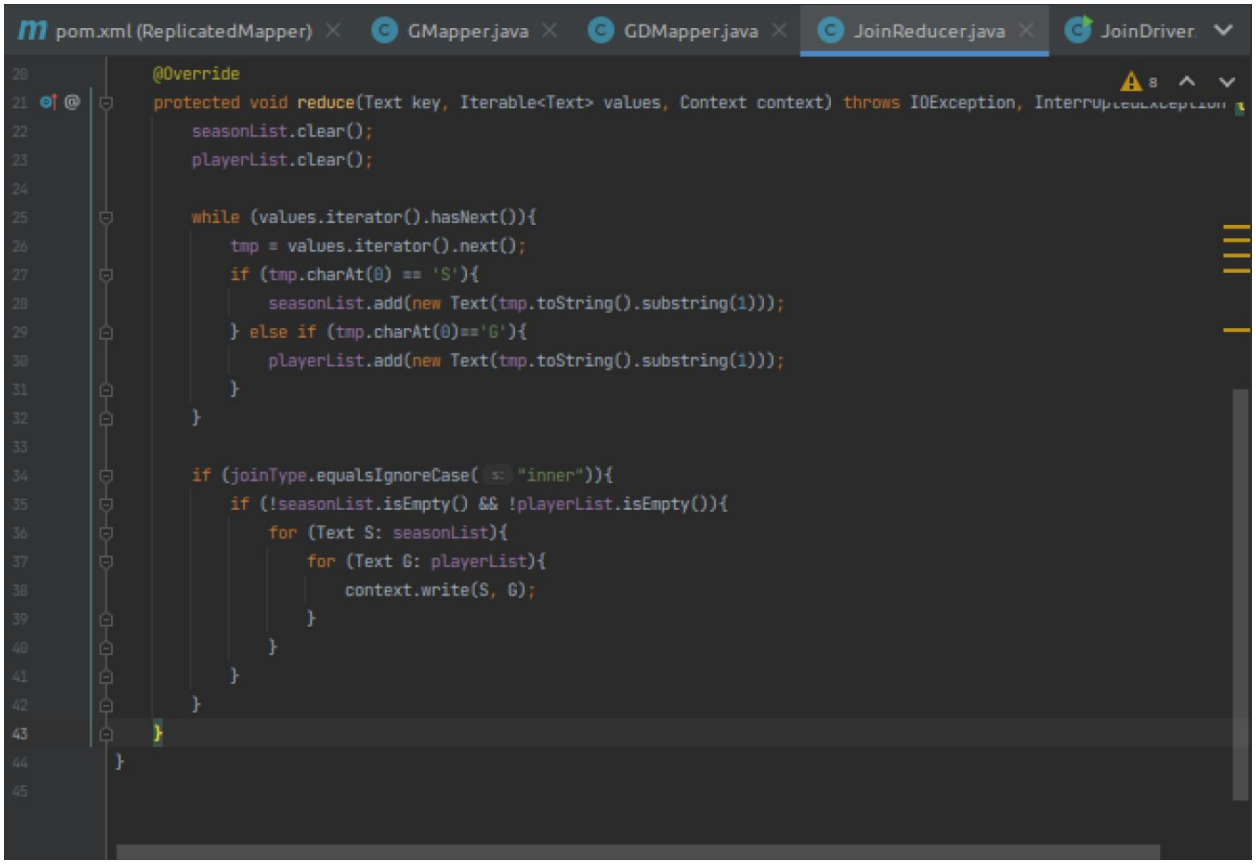
- Mapper



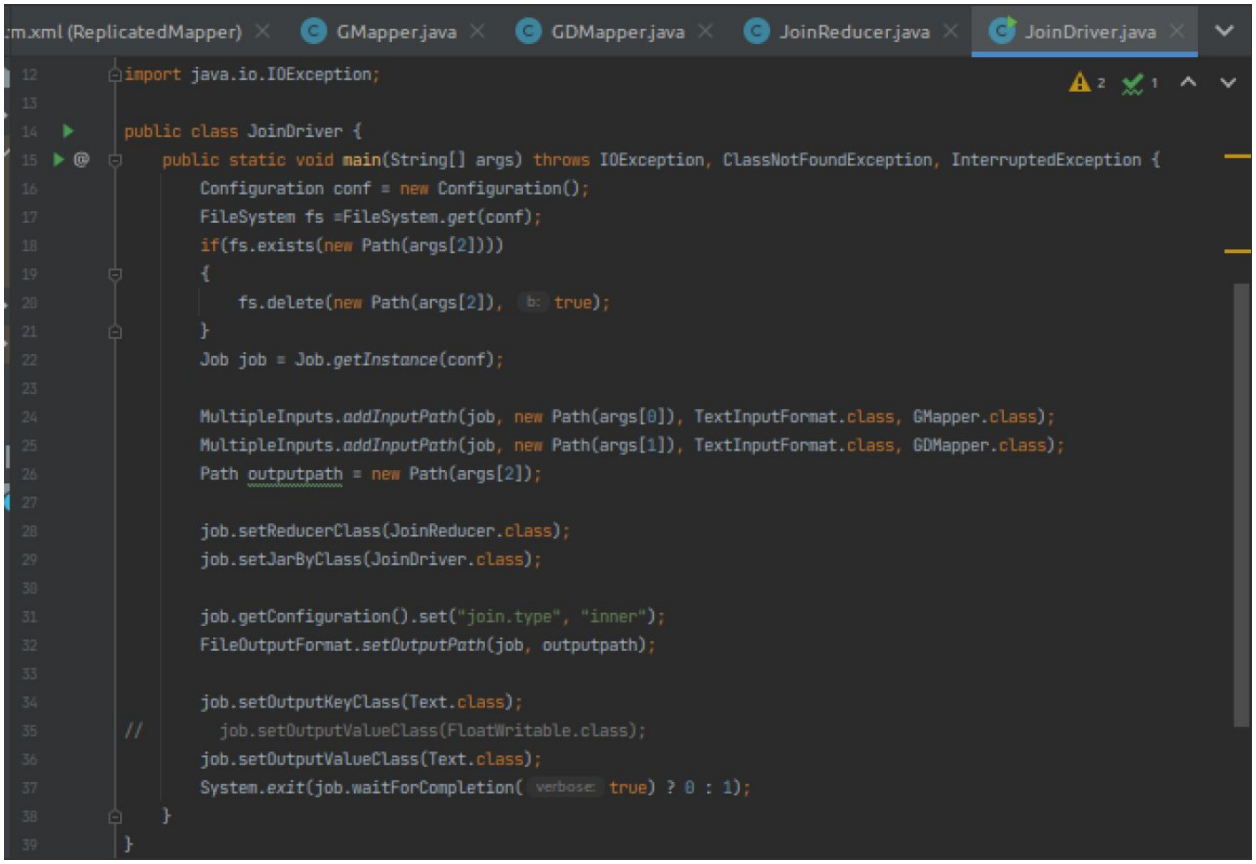

pom.xml (ReplicatedMapper)
GMapper.java
GDMapper.java
JoinReducer.java
JoinDriver
public class GDMapper extends Mapper<LongWritable, Text, Text, Text> {
private Text outkey = new Text();
private Text outval = new Text();
@Override
protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
String input = value.toString();
String[] Tokens = input.split(s: ",");
String gameId = Tokens[0];
String playerName = Tokens[5];
if (!gameId.isEmpty() && !playerName.isEmpty() && Tokens.length>=27 && !Tokens[0].equalsIgnoreCase(s: "GAME_I
float pt3 = Float.parseFloat(Tokens[12]);
float pt2 = Float.parseFloat(Tokens[9]) - Float.parseFloat(Tokens[12]);
float pt1 = Float.parseFloat(Tokens[15]);
float totalPts = pt1*1 + pt2*2 + pt3*3;
if (totalPts >= 40){
outkey.set(gameId);
outval.set("G"+playerName);
context.write(outkey, outval);
}
}
}

- Reducer


pom.xml (ReplicatedMapper)
GMapper.java
GDMapper.java
JoinReducer.java
JoinDriver
import java.util.ArrayList;
public class JoinReducer extends Reducer<Text, Text, Text, Text> {
private static final Text EMPTY_TEXT = new Text(string: "");
private Text tmp = new Text();
private ArrayList<Text> seasonList = new ArrayList<>();
private ArrayList<Text> playerList = new ArrayList<>();
private String joinType = null;
@Override
protected void setup(Context context) throws IOException, InterruptedException {
joinType = context.getConfiguration().get("join.type");
}
@Override
protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException {
seasonList.clear();
playerList.clear();
while (values.iterator().hasNext()){
tmp = values.iterator().next();
if (tmp.charAt(0) == 'S'){
seasonList.add(new Text(tmp.toString().substring(1)));
} else if (tmp.charAt(0)=='G'){
playerList.add(new Text(tmp.toString().substring(1)));
}
}


pom.xml (ReplicatedMapper)
GMapper.java
GDMapper.java
JoinReducer.java
JoinDriver
@Override
protected void reduce(Text key, Iterable<Text> values, Context context) throws IOException, InterruptedException
seasonList.clear();
playerList.clear();
while (values.iterator().hasNext()){
tmp = values.iterator().next();
if (tmp.charAt(0) == 'S'){
seasonList.add(new Text(tmp.toString().substring(1)));
} else if (tmp.charAt(0)=='G'){
playerList.add(new Text(tmp.toString().substring(1)));
}
}
if (joinType.equalsIgnoreCase(s: "inner")){
if (!seasonList.isEmpty() && !playerList.isEmpty()){
for (Text S: seasonList){
for (Text G: playerList){
context.write(S, G);
}
}
}
}
}

- Driver


m.xml (ReplicatedMapper)
GMapper.java
GDMapper.java
JoinReducer.java
JoinDriver.java
import java.io.IOException;
public class JoinDriver {
public static void main(String[] args) throws IOException, ClassNotFoundException, InterruptedException {
Configuration conf = new Configuration();
FileSystem fs =FileSystem.get(conf);
if(fs.exists(new Path(args[2])))
{
fs.delete(new Path(args[2]), b: true);
}
Job job = Job.getInstance(conf);
MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class, GMapper.class);
MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class, GDMapper.class);
Path outputpath = new Path(args[2]);
job.setReducerClass(JoinReducer.class);
job.setJarByClass(JoinDriver.class);
job.getConfiguration().set("join.type", "inner");
FileOutputFormat.setOutputPath(job, outputpath);
job.setOutputKeyClass(Text.class);
// job.setOutputValueClass(FloatWritable.class);
job.setOutputValueClass(Text.class);
System.exit(job.waitForCompletion(verbose: true) ? 0 : 1);
}
}

Output:

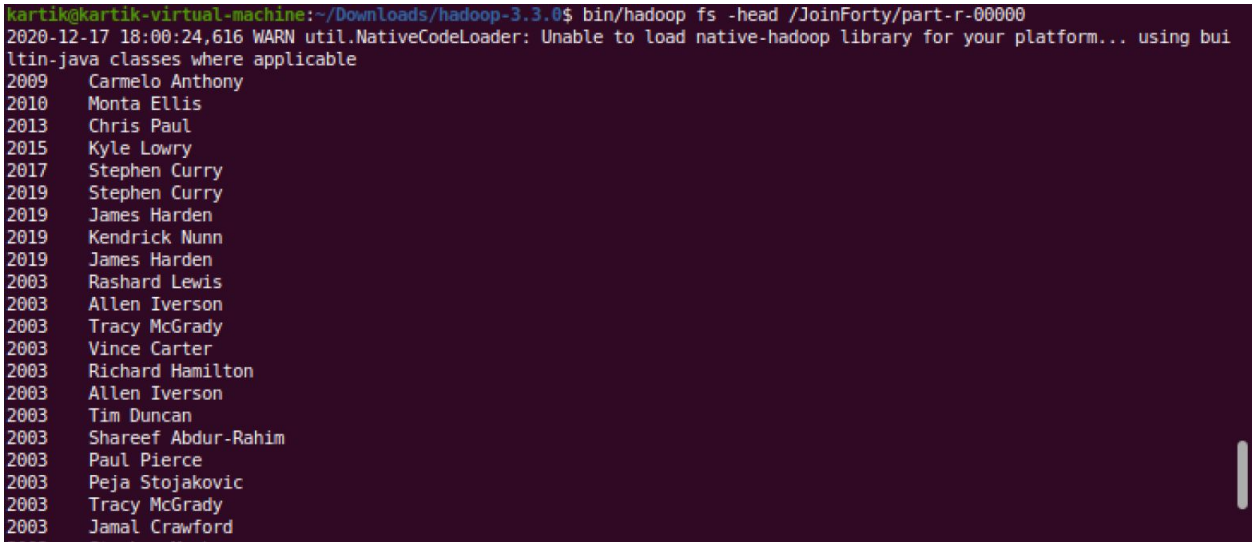
```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -head /JoinOutput/part-r-00000
2020-12-17 17:36:01,222 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
2003    Tariq Abdul-Wahad
2003    Travis Best
2003    Ademola Okulaja
2003    Andrei Kirilenko
2003    Ben Handlogten
2003    Carlos Arroyo
2003    Curtis Borchardt
2003    DeShawn Stevenson
2003    Demetrius Alexander
2003    Greg Ostertag
2003    Jarron Collins
2003    Jermaine Boyette
2003    Keon Clark
2003    Lavor Postell
2003    Matt Harpring
2003    Michael Ruffin
2003    Mo Williams
2003    Paul Grant
2003    Raja Bell
2003    Raul Lopez
2003    Sasha Pavlovic
2003    Antawn Jamison
2003    Danny Fortson
2003    Dirk Nowitzki
2003    Eduardo Najera
2003    Jiri Welsch
2003    Josh Howard
2003    Raef LaFrentz
2003    Shawn Bradley
2003    Steve Nash
2003    Desmond Mason
2003    Daniel Santiago
2003    Damon Jones
2003    Brian Skinner
2003    Antonio Meeking
```

What good is a join alone? We need to analyse the join result!
I have used a code from our Assignment 4 using ***KeyValueInputFormat*** and calculated the count players in every season. Here's the result:

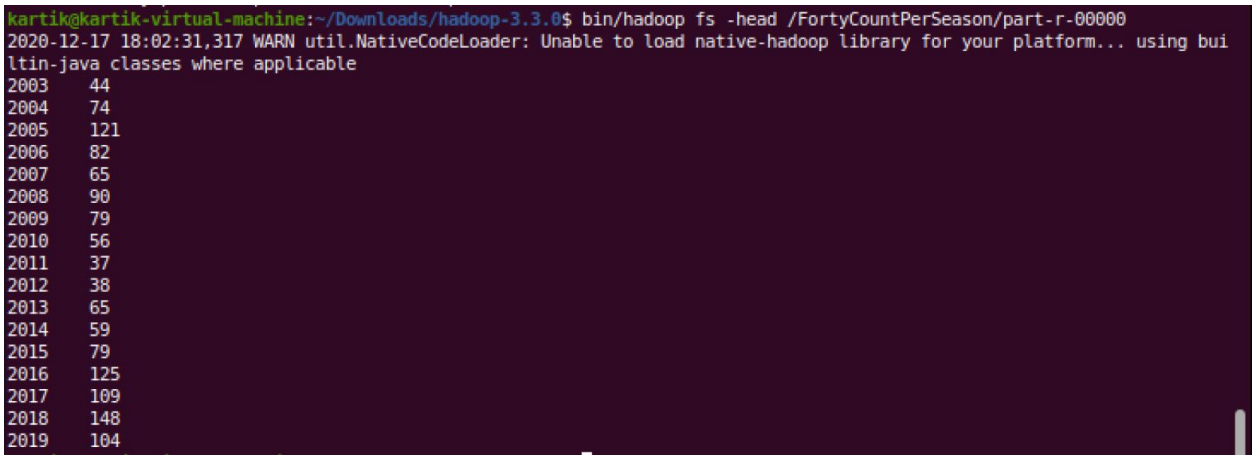
```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop jar /home/kartik/IdeaProjects/A4-KeyValTextInputForm
at/out/artifacts/A4_KeyValTextInputFormat_jar/A4-KeyValTextInputFormat.jar kvDriver /JoinOutput/part-r-00000 /PlayerCou
ntPerSeason
```

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop fs -head /PlayerCountPerSeason/part-r-00000
2020-12-17 17:43:20,670 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
2003    30703
2004    32601
2005    34521
2006    34144
2007    33917
2008    34140
2009    34175
2010    34091
2011    28391
2012    36198
2013    36386
2014    37244
2015    37160
2016    36978
2017    35719
2018    35348
2019    25066
```

A small modification in the above code, allowed me to join only the players who had a 40 point game. Below are the players who scored a 40 point game with the season in which they had that 40 point game:



I have used the above join output to calculate the total number of players that have scored 40+ in a game, per season. Below is the result:



As seen above, there are more players scoring 40 points in game since 2016. Atleast a 100 players have had 40 point games. This again points to how the league is progressing with data analysis and how players are improving their performances with the aid of technology.

Some analysis on HIVE (mostly failed analysis :( ) :

Load the table Games_details.csv from hdfs

```
hive> CREATE EXTERNAL TABLE gameDetails(`GAME_ID` String, `TEAM_ID` String, `TEAM_ABBREVIATION` String, `TEAM_CITY` Str
ing, `PLAYER_ID` String, `PLAYER_NAME` String, `START_POSITION` String, `COMMENT` String, `MIN` String, `FGM` Float, `F
GA` Float, `FG_PCT` Float, `FG3M` Float, `FG3A` Float, `FG3_PCT` Float, `FTM` Float, `FTA` Float, `FT_PCT` Float, `OREB
` Float, `DREB` Float, `AST` Float, `STL` Float, `BLK` Float, `TO` Float, `PF` Float, `PTS` Float, `PLUS_MINUS` Float)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' tblproperties("skip.header.line.count"="1");
FAILED: ParseException line 1:515 missing EOF at ',' near ''''
hive> CREATE EXTERNAL TABLE gameDetails(`GAME_ID` String, `TEAM_ID` String, `TEAM_ABBREVIATION` String, `TEAM_CITY` Str
ing, `PLAYER_ID` String, `PLAYER_NAME` String, `START_POSITION` String, `COMMENT` String, `MIN` String, `FGM` Float, `F
GA` Float, `FG_PCT` Float, `FG3M` Float, `FG3A` Float, `FG3_PCT` Float, `FTM` Float, `FTA` Float, `FT_PCT` Float, `OREB
` Float, `DREB` Float, `AST` Float, `STL` Float, `BLK` Float, `TO` Float, `PF` Float, `PTS` Float, `PLUS_MINUS` Float)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' tblproperties("skip.header.line.count"="1");
FAILED: ParseException line 1:515 missing EOF at ',' near ''''
hive> CREATE EXTERNAL TABLE gameDetails(`GAME_ID` String, `TEAM_ID` String, `TEAM_ABBREVIATION` String, `TEAM_CITY` Str
ing, `PLAYER_ID` String, `PLAYER_NAME` String, `START_POSITION` String, `COMMENT` String, `MIN` String, `FGM` Float, `F
GA` Float, `FG_PCT` Float, `FG3M` Float, `FG3A` Float, `FG3_PCT` Float, `FTM` Float, `FTA` Float, `FT_PCT` Float, `OREB
` Float, `DREB` Float, `AST` Float, `STL` Float, `BLK` Float, `TO` Float, `PF` Float, `PTS` Float, `PLUS_MINUS` Float)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' tblproperties('skip.header.line.count'='1');
FAILED: ParseException line 1:515 missing EOF at ',' near ''''
hive> CREATE EXTERNAL TABLE gameDetails(`GAME_ID` String, `TEAM_ID` String, `TEAM_ABBREVIATION` String, `TEAM_CITY` Str
ing, `PLAYER_ID` String, `PLAYER_NAME` String, `START_POSITION` String, `COMMENT` String, `MIN` String, `FGM` Float, `F
GA` Float, `FG_PCT` Float, `FG3M` Float, `FG3A` Float, `FG3_PCT` Float, `FTM` Float, `FTA` Float, `FT_PCT` Float, `OREB
` Float, `DREB` Float, `AST` Float, `STL` Float, `BLK` Float, `TO` Float, `PF` Float, `PTS` Float, `PLUS_MINUS` Float)
ROW FORMAT DELIMITED FIELDS TERMINATED BY ',' LINES TERMINATED BY '\n' tblproperties('skip.header.line.count'='1');
OK
Time taken: 0.065 seconds
```

```
hive> LOAD DATA INPATH 'hdfs://localhost:9000/NBA/games_details.csv' OVERWRITE INTO TABLE gameDetails;
Loading data to table default.gamedetails
OK
Time taken: 0.416 seconds
hive> describe gameDetails;
OK
game_id                 string
team_id                 string
team_abbreviation       string
team_city               string
player_id               string
player_name             string
start_position          string
comment                 string
min                     string
fgm                     float
fga                     float
fg_pct                  float
fg3m                    float
fg3a                    float
fg3_pct                 float
ftm                     float
fta                     float
ft_pct                  float
oreb                    float
dreb                    float
ast                     float
stl                     float
blk                     float
to                      float
pf                      float
pts                     float
plus_minus              float
Time taken: 0.06 seconds, Fetched: 27 row(s)
```

```
hive> select PLAYER_NAME from gameDetails where pts >=40 and TEAM_ABBREVIATION="CEL";
OK
Time taken: 0.121 seconds
hive> select GAME_ID,PLAYER_NAME from gameDetails where pts >=40 and TEAM_ABBREVIATION="CEL";
OK
Time taken: 0.11 seconds
hive> select * from gameDetails where pts >=40 and TEAM_ABBREVIATION="CEL";
OK
```

```
Kill Command = /home/kartik/Downloads/hadoop-3.3.0//bin/mapred job  -kill job_1608324078719_0001
Hadoop job information for Stage-1: number of mappers: 0; number of reducers: 0
2020-12-18 12:44:29,838 Stage-1 map = 0%,  reduce = 0%
Ended Job = job_1608324078719_0001 with errors
Error during job, obtaining debugging information...
FAILED: Execution Error, return code 2 from org.apache.hadoop.hive.ql.exec.mr.MapRedTask
MapReduce Jobs Launched:
Stage-Stage-1:  HDFS Read: 0 HDFS Write: 0 FAIL
Total MapReduce CPU Time Spent: 0 msec
hive>
```

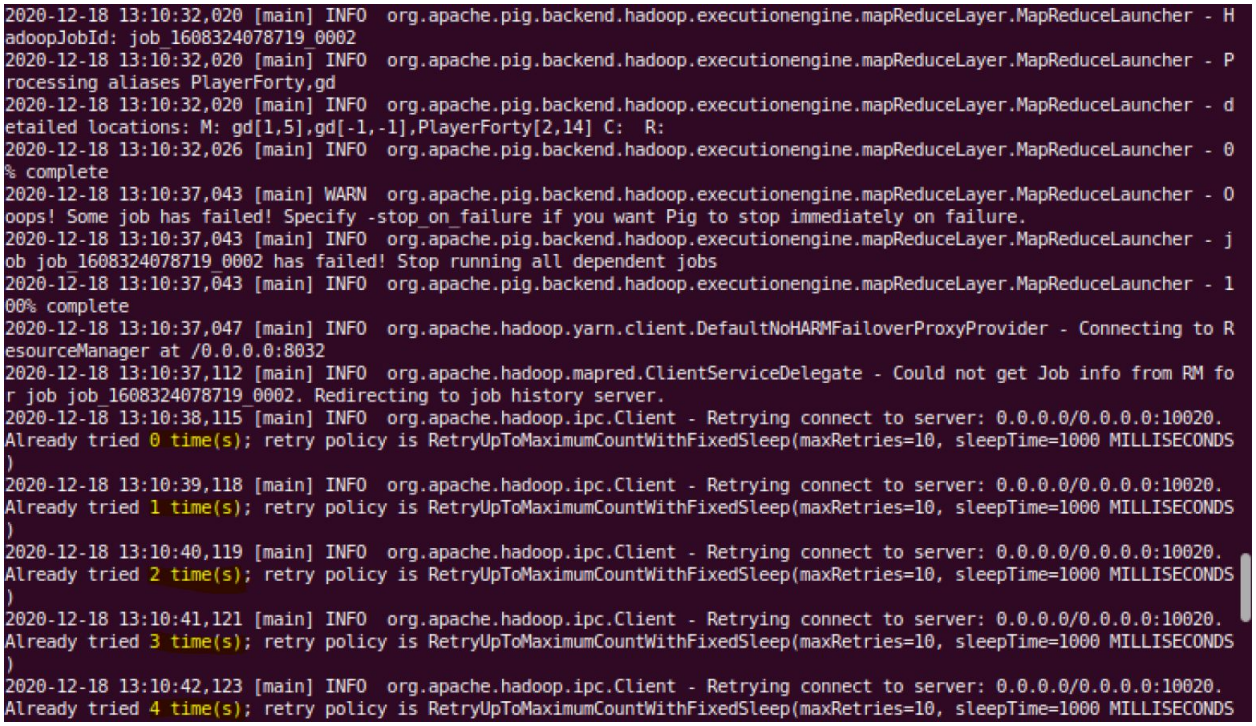
Local pc hung with this query (not usual of hive)

```
hive> Select START_POSITION, count(*) from gameDetails GROUP BY START_POSITION;
Query ID = kartik_20201218123304_f9166190-6698-4a79-99cd-df7d1d28a23b
Total jobs = 1
Launching Job 1 out of 1
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_1608316496685_0005, Tracking URL = http://kartik-virtual-machine:8088/proxy/application_160831649668
5_0005/
Kill Command = /home/kartik/Downloads/hadoop-3.3.0//bin/mapred job  -kill job_1608316496685_0005
```

Some analysis on Pig(again, mostly unsuccessful):

```
grunt> gd = LOAD '/NBA/games_details.csv' USING PigStorage(',') AS (GAME_ID:chararray, TEAM_ID:chararray, TEAM_ABBREVIA
TION:chararray, TEAM_CITY:chararray, PLAYER_ID:chararray, PLAYER_NAME:chararray, START_POSITION:chararray, COMMENT:char
array, MIN:chararray, FGM:float, FGA:float, FG_PCT:float, FG3M:float, FG3A:float, FG3_PCT:float, FTM:float, FTA:float,
FT_PCT:float, OREB:float, DREB:float, AST:float, STL:float, BLK:float, TO:float, PF:float, PTS:float, PLUS_MINUS:float)
;
grunt> PlayerForty = filter gd by PTS>=40;
2020-12-18 13:09:48,112 [main] WARN  org.apache.pig.newplan.BaseOperatorPlan - Encountered Warning IMPLICIT_CAST_TO_FLO
AT 1 time(s).
```

Getting stuck here:

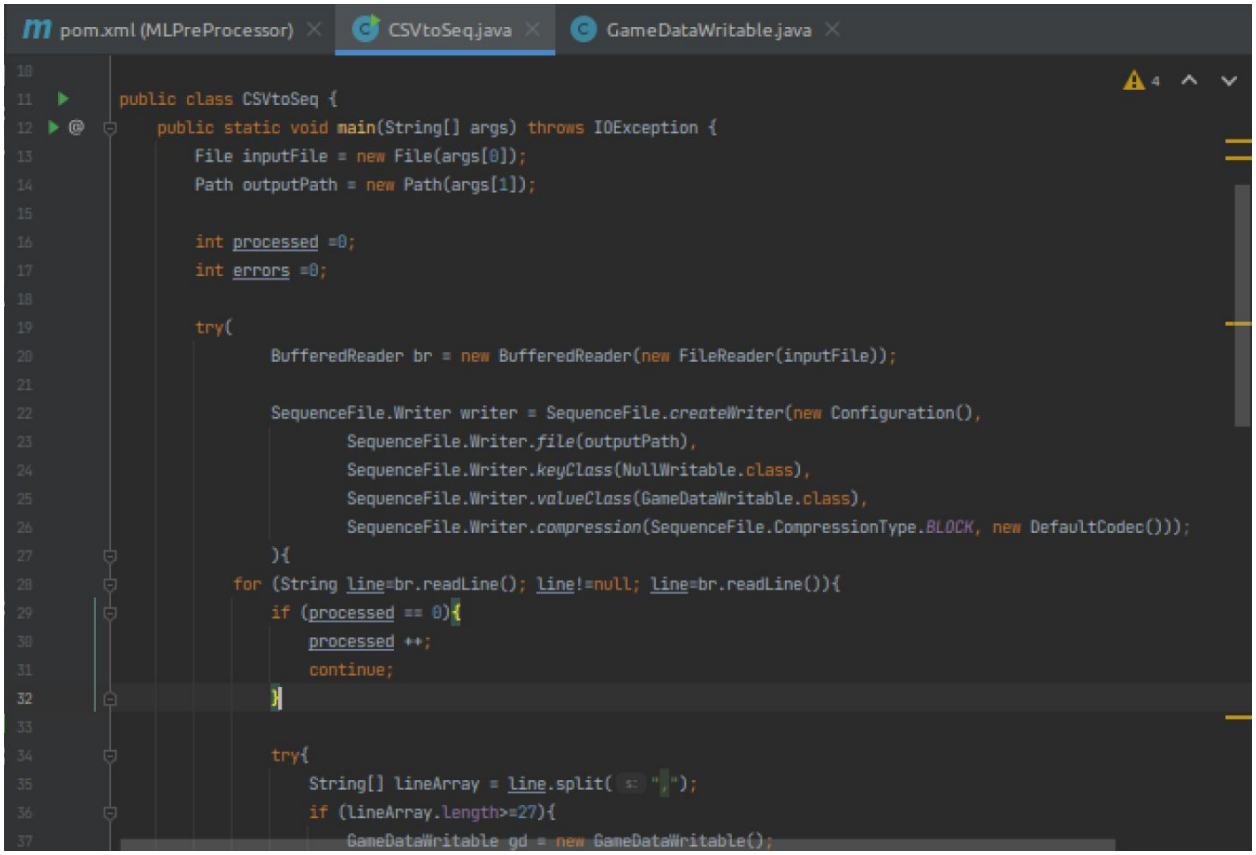


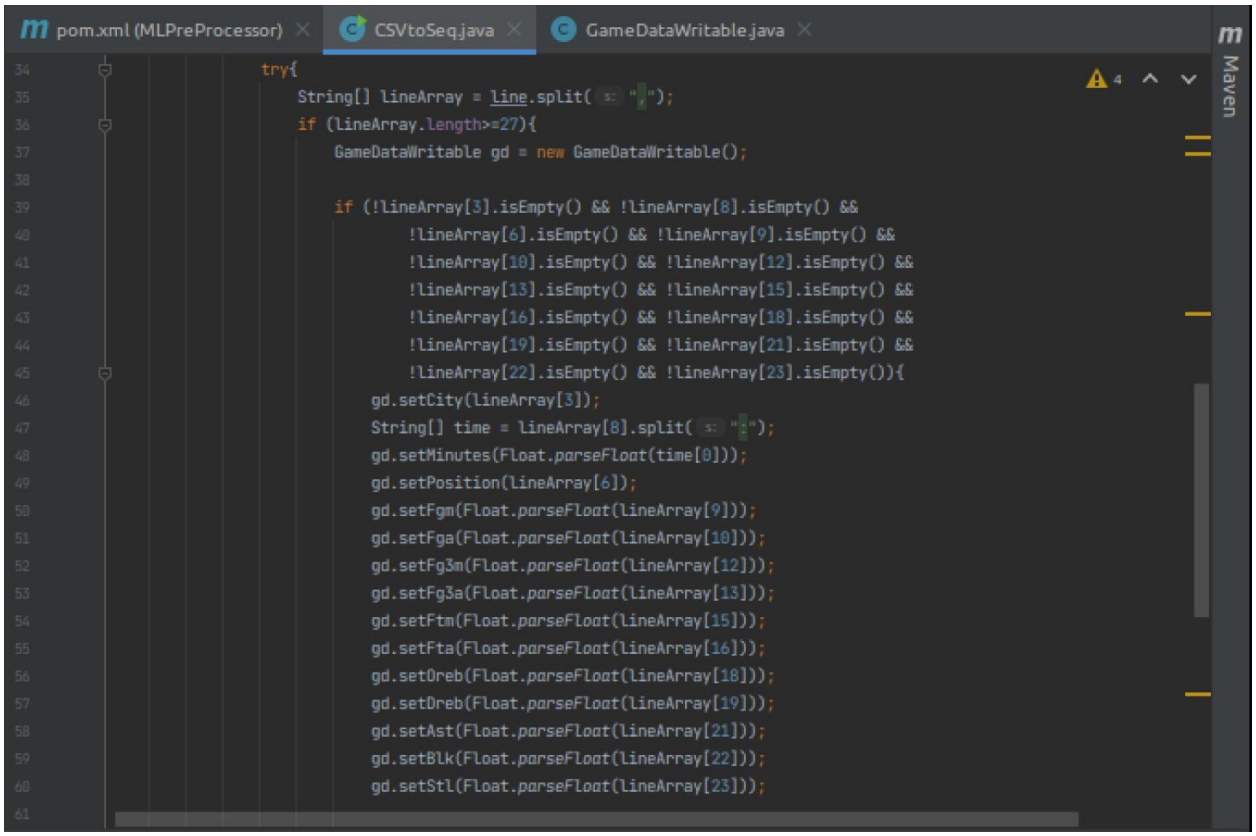
3. Advanced Analysis - Machine Learning

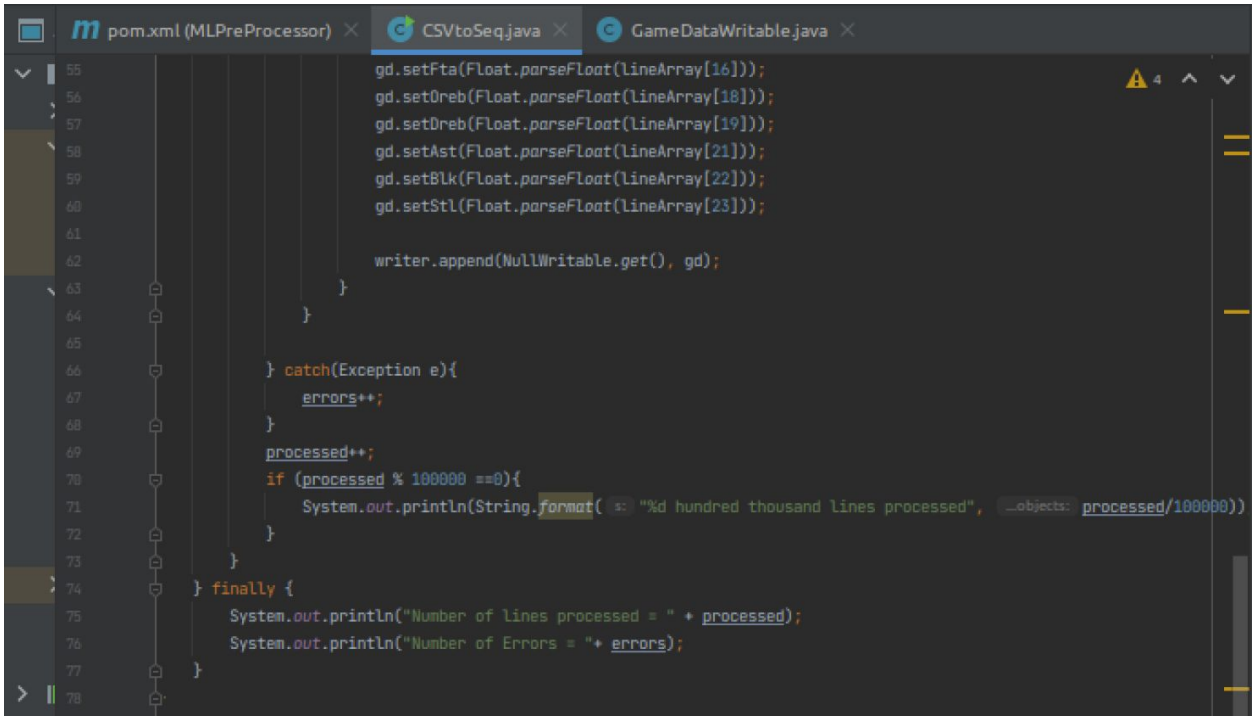
a) Mahout

I tried to convert CSV to Sequence File with preprocessed data and then tried to run Mahout classification algorithm with the Sequence file. Unfortunately, the code gave an error. Below is a snippet of the pre-processing conversion from CSV to Sequence File removing all empty values. Aim was to be able to predict a player's position based on other column values.

Code Snippet:




pom.xml (MLPreProcessor)
CSVtoSeq.java
GameDataWritable.java
Maven
try{
String[] lineArray = line.split(s: ",");
if (lineArray.length>=27){
GameDataWritable gd = new GameDataWritable();
if (!lineArray[3].isEmpty() && !lineArray[8].isEmpty() &&
!lineArray[6].isEmpty() && !lineArray[9].isEmpty() &&
!lineArray[10].isEmpty() && !lineArray[12].isEmpty() &&
!lineArray[13].isEmpty() && !lineArray[15].isEmpty() &&
!lineArray[16].isEmpty() && !lineArray[18].isEmpty() &&
!lineArray[19].isEmpty() && !lineArray[21].isEmpty() &&
!lineArray[22].isEmpty() && !lineArray[23].isEmpty()){
gd.setCity(lineArray[3]);
String[] time = lineArray[8].split(s: ":");
gd.setMinutes(Float.parseFloat(time[0]));
gd.setPosition(lineArray[6]);
gd.setFgm(Float.parseFloat(lineArray[9]));
gd.setFga(Float.parseFloat(lineArray[10]));
gd.setFg3m(Float.parseFloat(lineArray[12]));
gd.setFg3a(Float.parseFloat(lineArray[13]));
gd.setFtm(Float.parseFloat(lineArray[15]));
gd.setFta(Float.parseFloat(lineArray[16]));
gd.setOreb(Float.parseFloat(lineArray[18]));
gd.setDreb(Float.parseFloat(lineArray[19]));
gd.setAst(Float.parseFloat(lineArray[21]));
gd.setBlk(Float.parseFloat(lineArray[22]));
gd.setStl(Float.parseFloat(lineArray[23]));


pom.xml (MLPreProcessor)
CSVtoSeq.java
GameDataWritable.java
gd.setFta(Float.parseFloat(lineArray[16]));
gd.setOreb(Float.parseFloat(lineArray[18]));
gd.setDreb(Float.parseFloat(lineArray[19]));
gd.setAst(Float.parseFloat(lineArray[21]));
gd.setBlk(Float.parseFloat(lineArray[22]));
gd.setStl(Float.parseFloat(lineArray[23]));
writer.append(NullWritable.get(), gd);
}
}
} catch(Exception e){
errors++;
}
processed++;
if (processed % 100000 ==0){
System.out.println(String.format(s: "%d hundred thousand lines processed", ...objects: processed/100000));
}
}
} finally {
System.out.println("Number of lines processed = " + processed);
System.out.println("Number of Errors = "+ errors);
}

Output:

```
kartik@kartik-virtual-machine:~/Downloads/hadoop-3.3.0$ bin/hadoop jar /home/kartik/IdeaProjects/MLPreProcessor/out/art
ifacts/MLPreProcessor_jar/MLPreProcessor.jar CSVtoSeq ~/Downloads/NBA/games_details.csv /GDSeq
2020-12-17 22:45:16,196 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using bui
ltin-java classes where applicable
2020-12-17 22:45:16,928 INFO compress.CodecPool: Got brand-new compressor [.deflate]
1 hundred thousand lines processed
2 hundred thousand lines processed
3 hundred thousand lines processed
4 hundred thousand lines processed
5 hundred thousand lines processed
Number of lines processed = 576783
Number of Errors = 0
```

I successfully converted the csv file to sequence file but couldn't convert it further into a vector using the mahout library. Awaiting a good documentation on the most stable release (launched October 2020)

b) Python machine learning with Scikit-learn

After several attempts at implementing machine learning via Mahout, I decided to go back to the all-famous Python package Scikit-learn to help me implement machine learning with this data.

The data was stored in a Pandas dataframe and then fed into the Scikit-learn algorithms. Pandas dataframes can store data 100MB-1Gb files and Python provides easy functionality for visualizations, pre-processing and implementing an ML algorithm.

Below I have implemented 2 classification algorithms: KNN classifier and XGBoost Classifier. Both are supervised learning algorithms.

Data pre-processing:

Drop irrelevant columns and handle missing values
From industry knowledge, I know that IDs will not impact the position of the player, hence I have dropped them.
I have also dropper the total points column as that highly correlates with the other features.

PTS = FG3M*3 + (FG-FG3)*2 + FTM*1

```
#MACHINE LEARNING WITH SCIKIT LEARN

#Remove unnecessary columsn
#Handle/Remove missing values

gd = games_details.drop(columns = ['GAME_ID', 'TEAM_ID','TEAM_CITY', 'TEAM_ABBREVIATION', 'PLAYER_ID', 'PLAYER_NAME','COMMENT',

gd.head()
```

	START_POSITION	FGM	FGA	FG_PCT	FG3M	FG3A	FG3_PCT	FTM	FTA	FT_PCT	OREB	DREB	REB	AST	STL	BLK	TO	PF
0	F	3.0	11.0	0.273	2.0	7.0	0.286	0.0	0.0	0.000	4.0	4.0	8.0	2.0	2.0	0.0	0.0	0.0
1	F	17.0	28.0	0.607	1.0	4.0	0.250	6.0	7.0	0.857	2.0	18.0	20.0	6.0	1.0	0.0	3.0	2.0
2	C	4.0	11.0	0.364	1.0	5.0	0.200	7.0	9.0	0.778	2.0	5.0	7.0	0.0	0.0	3.0	0.0	2.0
3	G	1.0	5.0	0.200	0.0	3.0	0.000	0.0	0.0	0.000	1.0	6.0	7.0	5.0	0.0	1.0	2.0	0.0
4	G	2.0	8.0	0.250	0.0	1.0	0.000	0.0	0.0	0.000	1.0	0.0	1.0	2.0	1.0	0.0	3.0	2.0

```
gd.isnull().sum()
```

```
START_POSITION    355408
FGM                92261
FGA                92261
FG_PCT             92261
FG3M               92261
FG3A               92261
FG3_PCT            92261
FTM                92261
FTA                92261
FT_PCT             92261
OREB               92261
DREB               92261
REB                92261
AST                92261
STL                92261
BLK                92261
TO                 92261
PF                 92261
dtype: int64
```

```
gd = gd.dropna()
```

```
gd.isnull().sum()
```

```
START_POSITION    0
FGM               0
FGA               0
FG_PCT            0
```

There are multiple ways to handle missing values. Replacing with median, mode, mean, previous entry, next entry, etc. For this example, I have used the simplest way - drop all rows that have a missing value.
isnull().sum() shows missing values per column. I have brought that down to 0 for every column using dropna() method.

To train our model, we need to split the data. I have used sklearn library to do this and split with a 70-30 ratio. This ratio may vary.

```
gd['START_POSITION'].unique()
```

```
array(['F', 'C', 'G'], dtype=object)
```

```
y = gd['START_POSITION']
X = gd.drop(columns=['START_POSITION'])

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

KNN - K Nearest Neighbours:
Classifies based on nearest neighbours for a point

```
In [42]: from sklearn.neighbors import KNeighborsClassifier
         neigh = KNeighborsClassifier(n_neighbors=3)
         neigh.fit(X_train, y_train)
```

```
Out[42]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                              weights='uniform')
```

```
In [43]: neigh.predict(X_test)
```

```
Out[43]: array(['F', 'G', 'G', ..., 'G', 'G', 'F'], dtype=object)
```

```
In [44]: neigh.score(X_test, y_test)
```

```
Out[44]: 0.597563729992622
```

Model gives an accuracy of 59.7% on test data.

XGBClassifier:

```
In [45]: from xgboost import XGBClassifier
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import accuracy_score
```

```
In [46]: model = XGBClassifier()
         model.fit(X_train, y_train)
```

```
Out[46]: XGBClassifier(base_score=0.5, booster=None, colsample_bylevel=1,
                       colsample_bynode=1, colsample_bytree=1, gamma=0, gpu_id=-1,
                       importance_type='gain', interaction_constraints=None,
                       learning_rate=0.300000012, max_delta_step=0, max_depth=6,
                       min_child_weight=1, missing=nan, monotone_constraints=None,
                       n_estimators=100, n_jobs=0, num_parallel_tree=1,
                       objective='multi:softprob', random_state=0, reg_alpha=0,
                       reg_lambda=1, scale_pos_weight=None, subsample=1,
                       tree_method=None, validate_parameters=False, verbosity=None)
```

```
In [48]: y_pred = model.predict(X_test)
         y_pred
```

```
Out[48]: array(['C', 'G', 'G', ..., 'G', 'G', 'F'], dtype=object)
```

```
In [50]: accuracy = accuracy_score(y_test, y_pred)
         accuracy
```

```
Out[50]: 0.6767650911719091
```

```
In [52]: model1 = XGBClassifier(base_score=1, learning_rate=0.100000012)
         model1.fit(X_train, y_train)

         y_pred1 = model1.predict(X_test)

         accuracy = accuracy_score(y_test, y_pred1)
         print(accuracy)

         0.6794302320328851
```

XGB Classifier gives much better results than KNN. We are able to predict the position of a player based on the match stats with an accuracy of 67%. Pretty good!

The second model reduces the learning rate from default of 0.3 to 0.1. The accuracy improves, but not by much - only 0.2 % improvement. We can tune the other hyperparameters to get accuracy up to 75-80%.

Note, all accuracies are calculated on test data - which the model has not seen before. The model was trained on 70% of sampled data and tested with the remaining 30%. Scikit learns train_test_split allows random sampling.

ACKNOWLEDGEMENT

I thank Professor Yusuf and the TAs - Parth and Sarthak, for helping me throughout the duration of this course. I have developed a good understanding of Big Data technologies and was able to tackle small barriers I faced in my code implementations with their help. Professor Yusuf has explained all concepts extremely clearly in his slides and lectures.

APPENDIX

This section contains code of every analysis shown above in the report.

1) Exploratory analysis with MongoDB
Column: “GAME_DATE_EST”
Map -

```
function(){
        var date = String(this.GAME_DATE_EST);
        var arr1 = date.split(“-”);
        emit(arr1[0], 1);
}
```

Reduce -

```
function(key, values){
        var sum = 0;
        values.forEach((val)=>sum+=val);
        return sum;
}
```

Column: “SEASON”
Map -

```
function(){
        emit(this.SEASON, 1);
}
```

Reduce -

```
function(key, values){
        var sum = 0;
        values.forEach((val)=>sum+=val);
        return sum;
}
```

2) Exploratory analysis on Games.csv
Mapper:

```
import jdk.nashorn.internal.parser.Token;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
```

```
import java.io.IOException;

public class GameMapper extends Mapper<LongWritable, Text, Text, HomeAwayStore> {

	private Text outkey = new Text();
//	private FloatWritable outval = new FloatWritable();
	private HomeAwayStore outval = new HomeAwayStore();

	@Override
	protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
	String input = value.toString();
	String Tokens[] = input.split(",");

	String pts_home = Tokens[7];
	String pts_away = Tokens[14];
	String ast_home = Tokens[11];
	String ast_away = Tokens[18];
	String reb_home = Tokens[12];
	String reb_away = Tokens[19];

	if ((!Tokens[0].equalsIgnoreCase("GAME_DATE_EST")) &&
		((pts_home!= null) && (!pts_home.isEmpty()) && (pts_away!= null) &&
(!pts_away.isEmpty())) &&
		((ast_home!= null) && (!ast_home.isEmpty()) && (ast_away!= null) &&
(!ast_away.isEmpty()))&&
		((reb_home!= null) && (!reb_home.isEmpty()) && (reb_away!= null) &&
(!reb_away.isEmpty()))
	){
	float home_pts = Float.valueOf(pts_home).floatValue();
	float away_pts = Float.valueOf(pts_away).floatValue();
	float home_ast = Float.valueOf(ast_home).floatValue();
	float away_ast = Float.valueOf(ast_away).floatValue();
	float home_reb = Float.valueOf(reb_home).floatValue();
	float away_reb = Float.valueOf(reb_away).floatValue();

	outval.setPts_diff(home_pts-away_pts);
	outval.setAst_diff(home_ast-away_ast);
	outval.setReb_diff(home_reb-away_reb);

	outkey.set(Tokens[5]);
	context.write(outkey, outval);
	}
```

```
//Count Home wins or season games
//		if ((!Tokens[0].equalsIgnoreCase("GAME_DATE_EST")) && !Tokens[7].equals("") &&
!Tokens[14].equals("")){
////			String season = Tokens[5];
//		float home_pts = Float.parseFloat(Tokens[7]);
//		float away_pts = Float.parseFloat(Tokens[14]);
//
//		if (home_pts>away_pts){
//			outkey.set("HomeWins");
//			outval.set(1);
//			context.write(outkey, outval);
//		}
////			outkey.set(season);
////			outval.set(1);
////			context.write(outkey, outval);
//		}



		}
}
```

Custom Writable:

```
import org.apache.hadoop.io.Writable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class HomeAwayStore implements Writable {
		private float pts_diff;
		private float ast_diff;
		private float reb_diff;

		public float getPts_diff() {
		return pts_diff;
		}

		public void setPts_diff(float pts_diff) {
		this.pts_diff = pts_diff;
		}
```

```
    public float getAst_diff() {
    return ast_diff;
    }

    public void setAst_diff(float ast_diff) {
    this.ast_diff = ast_diff;
    }

    public float getReb_diff() {
    return reb_diff;
    }

    public void setReb_diff(float reb_diff) {
    this.reb_diff = reb_diff;
    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
    pts_diff = dataInput.readFloat();
    ast_diff = dataInput.readFloat();
    reb_diff = dataInput.readFloat();
    }

    @Override
    public void write(DataOutput dataOutput) throws IOException {
    dataOutput.writeFloat(pts_diff);
    dataOutput.writeFloat(ast_diff);
    dataOutput.writeFloat(reb_diff);
    }

    @Override
    public String toString() {
    return "HomeAwayStore{" +
            "pts_diff=" + pts_diff +
            ", ast_diff=" + ast_diff +
            ", reb_diff=" + reb_diff +
            '}';
    }
}
```

Reducer:

```
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class GameReducer extends Reducer<Text, HomeAwayStore, Text, HomeAwayStore> {
//	private FloatWritable sumOfGames = new FloatWritable();
	private HomeAwayStore result= new HomeAwayStore();

	@Override
	protected void reduce(Text key, Iterable<HomeAwayStore> values, Context context)
throws IOException, InterruptedException {
	int count =0;
	float pts_diff_sum =0;
	float ast_diff_sum =0;
	float reb_diff_sum =0;

	for (HomeAwayStore val: values){
	pts_diff_sum += val.getPts_diff();
	ast_diff_sum += val.getAst_diff();
	reb_diff_sum += val.getReb_diff();
	count++;
	}

	result.setPts_diff(pts_diff_sum/count);
	result.setAst_diff(ast_diff_sum/count);
	result.setReb_diff(reb_diff_sum/count);

	context.write(key, result);


//	int count =0;
//	for (FloatWritable val: values){
//	count += val.get();
//	}
//	sumOfGames.set(count);
//	context.write(key, sumOfGames);
	}
}
```

Driver:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
```

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class GameDriver {
	public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
	Configuration conf = new Configuration();
	FileSystem fs =FileSystem.get(conf);
	if(fs.exists(new Path(args[1])))
	{
	fs.delete(new Path(args[1]), true);
	}
	Job job = Job.getInstance(conf);
	job.setMapperClass(GameMapper.class);
	job.setReducerClass(GameReducer.class);
//	job.setCombinerClass(GameReducer.class);
	job.setJarByClass(GameDriver.class);

	TextInputFormat.addInputPath(job, new Path(args[0]));
	TextOutputFormat.setOutputPath(job, new Path(args[1]));

	job.setInputFormatClass(TextInputFormat.class);
	job.setMapOutputKeyClass(Text.class);
//	job.setMapOutputValueClass(FloatWritable.class);
	job.setMapOutputValueClass(HomeAwayStore.class);
	job.setOutputKeyClass(Text.class);
//	job.setOutputValueClass(FloatWritable.class);
	job.setOutputValueClass(HomeAwayStore.class);
	System.exit(job.waitForCompletion(true) ? 0 : 1);
	}
}
```

3) Exploratory analysis on Game Details.csv

Mapper:

```
import org.apache.hadoop.io.FloatWritable;
```

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class GDMapper extends Mapper<LongWritable, Text, Text, FloatWritable> {
	private Text outkey = new Text();
	private FloatWritable outval = new FloatWritable();

	@Override
	protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
	String input = value.toString();
	String[] Tokens = input.split(",");

	if(!Tokens[0].equalsIgnoreCase("GAME_ID") && Tokens.length>=20){
//	String position = Tokens[6];
//	String len = String.valueOf(Tokens.length);
//	String[] time = Tokens[8].split(":");
	float fg_pct = Float.parseFloat(Tokens[11]);
//	int minutes = Integer.parseInt(time[0]);
	outkey.set(Tokens[2]);
	outval.set(fg_pct);
	context.write(outkey, outval);

	}


	}
}
```

Reducer:

```
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class GDReducer extends Reducer<Text, FloatWritable, Text, FloatWritable> {
	private FloatWritable result = new FloatWritable();
```

```
	@Override
	protected void reduce(Text key, Iterable<FloatWritable> values, Context context) throws
IOException, InterruptedException {
	float sum = 0;
	int count =0;
	for (FloatWritable val: values){
	sum+=val.get();
	count++;
	}
	result.set(sum/count);
	context.write(key, result);
	}
}
```

Driver:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class GDDriver {
	public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
	Configuration conf = new Configuration();
	FileSystem fs =FileSystem.get(conf);
	if(fs.exists(new Path(args[1])))
	{
	fs.delete(new Path(args[1]), true);
	}
	Job job = Job.getInstance(conf);
	job.setMapperClass(GDMapper.class);
	job.setReducerClass(GDReducer.class);
//	job.setCombinerClass(GDReducer.class);
	job.setJarByClass(GDDriver.class);
```

```
        FileInputFormat.setInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setInputFormatClass(TextInputFormat.class);
        job.setMapOutputKeyClass(Text.class);
//      job.setMapOutputValueClass(FloatWritable.class);
        job.setMapOutputValueClass(FloatWritable.class);
        job.setOutputKeyClass(Text.class);
//      job.setOutputValueClass(FloatWritable.class);
        job.setOutputValueClass(FloatWritable.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}
```

4) Exploratory analysis on Players.csv, Ranking.csv, Teams.csv - one common file for all 3

Mapper:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class PRTMapper extends Mapper<LongWritable, Text, Text, IntWritable> {
        private Text outkey = new Text();
        private IntWritable outval = new IntWritable();

        @Override
        protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        String input = value.toString();
        String[] Tokens = input.split(",");

        if(!Tokens[1].equalsIgnoreCase("TEAM_ID") & !Tokens[9].isEmpty()){
//      String conference = Tokens[4];
        String arena = Tokens[8];
        outkey.set(arena);
        outval.set(Integer.parseInt(Tokens[9]));
        context.write(outkey, outval);

        }
        }
```

```
}
```

Reducer:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.VIntWritable;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class PRTReducer extends Reducer<Text, IntWritable, Text, IntWritable> {
        private IntWritable result = new IntWritable();

        @Override
        protected void reduce(Text key, Iterable<IntWritable> values, Context context) throws
IOException, InterruptedException {
        int sum = 0;
        for (IntWritable val: values){
        sum += val.get();
        }
        result.set(sum);

        context.write(key, result);
        }
}
```

Driver:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class PRTDriver {
        public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
        Configuration conf = new Configuration();
```

```
        FileSystem fs =FileSystem.get(conf);
        if(fs.exists(new Path(args[1])))
        {
        fs.delete(new Path(args[1]), true);
        }
        Job job = Job.getInstance(conf);
        job.setMapperClass(PRTMapper.class);
        job.setReducerClass(PRTReducer.class);
        job.setCombinerClass(PRTReducer.class);
        job.setJarByClass(PRTDriver.class);

        TextInputFormat.addInputPath(job, new Path(args[0]));
        TextOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setInputFormatClass(TextInputFormat.class);
        job.setMapOutputKeyClass(Text.class);
//      job.setMapOutputValueClass(FloatWritable.class);
        job.setMapOutputValueClass(IntWritable.class);
        job.setOutputKeyClass(Text.class);
//      job.setOutputValueClass(FloatWritable.class);
        job.setOutputValueClass(IntWritable.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}
```

5) Intermediate analysis - Secondary Sort on Games_Details.csv

```
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class SSMapper extends Mapper<LongWritable, Text, PlayerMinutePair, FloatWritable> {
        private PlayerMinutePair outkey = new PlayerMinutePair();
        private FloatWritable outval = new FloatWritable();

        @Override
        protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        String input = value.toString();
        String[] Tokens = input.split(",");
```

```
        if (!Tokens[0].equalsIgnoreCase("GAME_ID") && Tokens.length>=20){
        String name = Tokens[5];
        float pt3 = Float.parseFloat(Tokens[12]);
        float pt2 = Float.parseFloat(Tokens[9]) - Float.parseFloat(Tokens[12]);
        float pt1 = Float.parseFloat(Tokens[15]);

        float ptTotal = pt1*1 + pt2*2 + pt3*3;

        String[] time = Tokens[8].split(":");
        float minutes = Float.parseFloat(time[0]);

        outkey.set(name,minutes);
        outval.set(ptTotal);

        context.write(outkey, outval);
        }
        }
}
```

SortComparator:

```
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

public class KeyComparator extends WritableComparator {

        public KeyComparator(){
        super(PlayerMinutePair.class, true);
        }

        @Override
        public int compare(WritableComparable a, WritableComparable b) {
        PlayerMinutePair key1 = (PlayerMinutePair) a;
        PlayerMinutePair key2 = (PlayerMinutePair) b;

        int nameCmp =
key1.playerName.toLowerCase().compareTo(key2.playerName.toLowerCase());
        if(nameCmp!=0){
        return nameCmp;
        }
        else{
        return -1*Float.compare(key1.minutes, key2.minutes);
        }
        }
```

```
}
```

Custom Writable:

```
import org.apache.hadoop.io.WritableComparable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;
import java.util.Locale;

public class PlayerMinutePair implements WritableComparable<PlayerMinutePair> {
        public String playerName;
        public float minutes;

        public PlayerMinutePair(){}

        public PlayerMinutePair(String name, float min){
        super();
        this.set(name, min);
        }

        public void set(String name, float min){
        this.playerName = (name==null)?"" : name;
        this.minutes = min;
        }

        @Override
        public void write(DataOutput dataOutput) throws IOException {
        dataOutput.writeUTF(playerName);
        dataOutput.writeFloat(minutes);
        }

        @Override
        public void readFields(DataInput dataInput) throws IOException {
        this.playerName = dataInput.readUTF();
        this.minutes = dataInput.readFloat();
        }

        @Override
        public int compareTo(PlayerMinutePair playerMinutePair) {
        int nameCmp =
playerName.toLowerCase().compareTo(playerMinutePair.playerName.toLowerCase());
        if (nameCmp!=0){
        return nameCmp;
```

```
        }
        else{
        return -1*Float.compare(minutes,playerMinutePair.minutes);
        }
        }
}
```

NaturalKey Partitioner:

```
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.mapreduce.Partitioner;

public class NaturalKeyPartitioner extends Partitioner<PlayerMinutePair, FloatWritable> {
        @Override
        public int getPartition(PlayerMinutePair playerMinutePair, FloatWritable floatWritable, int
i) {
        return Math.abs(playerMinutePair.playerName.hashCode() & Integer.MAX_VALUE) % i;
        }
}
```

NaturalKey Comparator:

```
import org.apache.hadoop.io.WritableComparable;
import org.apache.hadoop.io.WritableComparator;

import java.util.Locale;

public class NaturalKeyComparator extends WritableComparator {
        public NaturalKeyComparator(){
        super(PlayerMinutePair.class, true);
        }

        @Override
        public int compare(WritableComparable a, WritableComparable b) {
        PlayerMinutePair key1 = (PlayerMinutePair) a;
        PlayerMinutePair key2 = (PlayerMinutePair) b;

        return key1.playerName.toLowerCase().compareTo(key2.playerName.toLowerCase());
        }
}
```

Reducer:

```
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class SSReducer extends Reducer<PlayerMinutePair, FloatWritable, Text, Text> {
	private Text outkey = new Text();
	private Text outval = new Text();

	@Override
	protected void reduce(PlayerMinutePair key, Iterable<FloatWritable> values, Context
context) throws IOException, InterruptedException {

	for (FloatWritable val: values){
	String name = key.playerName;
	float minutes = key.minutes;
	float points = val.get();
	String ok = name + " | " + String.valueOf(minutes);
	outkey.set(ok);
	outval.set(String.valueOf(points));
	context.write(outkey, outval);
	}


	}
}
```

Driver:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class SSDriver {
```

```
	public static void main(String[] args) throws InterruptedException, IOException,
ClassNotFoundException {
	Configuration conf = new Configuration();
	FileSystem fs =FileSystem.get(conf);
	if(fs.exists(new Path(args[1])))
	{
	fs.delete(new Path(args[1]), true);
	}
	Job job = Job.getInstance(conf);
	job.setMapperClass(SSMapper.class);
	job.setReducerClass(SSReducer.class);
	job.setJarByClass(SSDriver.class);

	job.setPartitionerClass(NaturalKeyPartitioner.class);
	job.setGroupingComparatorClass(NaturalKeyComparator.class);
	job.setSortComparatorClass(KeyComparator.class);

	job.setNumReduceTasks(10);

	TextInputFormat.addInputPath(job, new Path(args[0]));
	TextOutputFormat.setOutputPath(job, new Path(args[1]));

	job.setInputFormatClass(TextInputFormat.class);
	job.setMapOutputKeyClass(PlayerMinutePair.class);
	job.setMapOutputValueClass(FloatWritable.class);
	job.setOutputKeyClass(Text.class);
//	job.setOutputValueClass(FloatWritable.class);
	job.setOutputValueClass(Text.class);
	System.exit(job.waitForCompletion(true) ? 0 : 1);
	}
}
```

6) Intermediate analysis with Median & Std calculation
Mapper:

```
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class MSMapper extends Mapper<LongWritable, Text, Text, FloatWritable> {
```

```
	private Text outkey = new Text();
	private FloatWritable outval = new FloatWritable();

	@Override
	protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
	String input = value.toString();
	String[] Tokens = input.split(",");

	if (!Tokens[1].equalsIgnoreCase("GAME_ID") && Tokens.length>=19 &&
!Tokens[15].isEmpty()){
	outkey.set(Tokens[5]);
	float fgAway = Float.parseFloat(Tokens[15]);
	outval.set(fgAway);
	context.write(outkey, outval);
	}
	}
}
```

Custom Writable:

```
import org.apache.hadoop.io.Writable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class MedStdTup implements Writable {

	private float median;
	private float stdDev;

	public float getStdDev() {
	return stdDev;
	}

	public void setStdDev(float stdDev) {
	this.stdDev = stdDev;
	}

	public float getMedian() {
	return median;
	}
```

```
	public void setMedian(float median) {
	this.median = median;
	}

	@Override
	public void write(DataOutput dataOutput) throws IOException {
	dataOutput.writeFloat(median);
	dataOutput.writeFloat(stdDev);
	}

	@Override
	public void readFields(DataInput dataInput) throws IOException {
	median = dataInput.readFloat();
	stdDev = dataInput.readFloat();
	}

	@Override
	public String toString() {
	return "MedStdTup{" +
		"median=" + median +
		", stdDev=" + stdDev +
		'}';
	}
}
```

Reducer:

```
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;

public class MSReducer extends Reducer<Text, FloatWritable, Text, MedStdTup> {

	private MedStdTup result = new MedStdTup();
	private ArrayList<Float> away_pct_list = new ArrayList<Float>();

	@Override
	protected void reduce(Text key, Iterable<FloatWritable> values, Context context) throws
IOException, InterruptedException {
	float sum = 0;
```

```
        float count =0;

        away_pct_list.clear();

        result.setStdDev(0);

        for (FloatWritable val: values){
        away_pct_list.add(val.get());
        sum += val.get();
        ++count;
        }

        Collections.sort(away_pct_list);

        if (count % 2 == 0){
        result.setMedian((away_pct_list.get((int) count/2 -1) + away_pct_list.get((int) count/2)) /
2.0f);
        } else {
        result.setMedian(away_pct_list.get((int) count/2));
        }


        float mean = sum / count;
        float sumOfSquares = 0.0f;

        for (Float f: away_pct_list){
        sumOfSquares += (f - mean)* (f - mean);
        }

        result.setStdDev((float) Math.sqrt(sumOfSquares / (count-1)));
        context.write(key, result);
        }
}
```

Driver:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

```
import java.io.IOException;

public class MSDriver {
	public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
	Configuration conf = new Configuration();
	FileSystem fs =FileSystem.get(conf);
	if(fs.exists(new Path(args[1])))
	{
	fs.delete(new Path(args[1]), true);
	}
	Job job = Job.getInstance(conf);
	job.setMapperClass(MSMapper.class);
	job.setReducerClass(MSReducer.class);
//	job.setCombinerClass(GameReducer.class);
	job.setJarByClass(MSDriver.class);

	TextInputFormat.addInputPath(job, new Path(args[0]));
	TextOutputFormat.setOutputPath(job, new Path(args[1]));

	job.setInputFormatClass(TextInputFormat.class);
	job.setMapOutputKeyClass(Text.class);
//	job.setMapOutputValueClass(FloatWritable.class);
	job.setMapOutputValueClass(FloatWritable.class);
	job.setOutputKeyClass(Text.class);
//	job.setOutputValueClass(FloatWritable.class);
	job.setOutputValueClass(MedStdTup.class);
	System.exit(job.waitForCompletion(true) ? 0 : 1);
	}
}
```

7) Intermediate analysis with Bloom Filter

a) Filter Training

Custom Writable:

```
import org.apache.hadoop.io.Writable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class GDWritable implements Writable {
```

```
    private String gameId;
    private String playerName;
    private float totalPts;

    @Override
    public void write(DataOutput dataOutput) throws IOException {
    dataOutput.writeUTF(gameId);
    dataOutput.writeUTF(playerName);
    dataOutput.writeFloat(totalPts);
    }

    @Override
    public void readFields(DataInput dataInput) throws IOException {
    gameId = dataInput.readUTF();
    playerName = dataInput.readUTF();
    totalPts = dataInput.readFloat();
    }

    public String getGameId() {
    return gameId;
    }

    public void setGameId(String gameId) {
    this.gameId = gameId;
    }

    public String getPlayerName() {
    return playerName;
    }

    public void setPlayerName(String playerName) {
    this.playerName = playerName;
    }

    public float getTotalPts() {
    return totalPts;
    }

    public void setTotalPts(float totalPts) {
    this.totalPts = totalPts;
    }
}
```

Driver with Mapper & Reducer:

```
import org.apache.commons.lang3.ObjectUtils;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.NullOutputFormat;
import org.apache.hadoop.util.bloom.BloomFilter;
import org.apache.hadoop.util.bloom.Key;
import org.apache.hadoop.util.hash.Hash;
import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.zip.GZIPInputStream;

public class BFDriver {

	public static class BFMapper extends Mapper<LongWritable, Text, NullWritable,
BloomFilter>{
	private BloomFilter filter = new BloomFilter(1_000_000,7, Hash.MURMUR_HASH);
	private int count = 0;

	@Override
	protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
	String input = value.toString();
	String[] Tokens = input.split(",");

	if (!Tokens[0].equalsIgnoreCase("GAME_ID") && Tokens.length>=20){
		String name = Tokens[5];
```

```
                float pt3 = Float.parseFloat(Tokens[12]);
                float pt2 = Float.parseFloat(Tokens[9]) - Float.parseFloat(Tokens[12]);
                float pt1 = Float.parseFloat(Tokens[15]);

                float ptTotal = pt1*1 + pt2*2 + pt3*3;

                if (ptTotal >= 40){
                Key filterKey = new Key(name.getBytes());
                filter.add(filterKey);
                count++;
                }
        }
        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
        context.write(NullWritable.get(), filter);
        }

        }


        public static class BFReducer extends Reducer<NullWritable, BloomFilter, NullWritable,
NullWritable>{

        private static String FILTER_OUTPUT_FILE_CONF = "bloomfilter.output.file";

        private BloomFilter filter = new BloomFilter(1_000_000,7, Hash.MURMUR_HASH);

        @Override
        protected void reduce(NullWritable key, Iterable<BloomFilter> values, Context context)
throws IOException, InterruptedException {
        for (BloomFilter val: values){
                filter.or(val);
        }
        }

        @Override
        protected void cleanup(Context context) throws IOException, InterruptedException {
        Path outputFilePath = new
Path(context.getConfiguration().get(FILTER_OUTPUT_FILE_CONF));
        FileSystem fs = FileSystem.get(context.getConfiguration());

        try (FSDataOutputStream fsdos = fs.create(outputFilePath)){
```

```
                filter.write(fsdos);
        } catch (Exception e){
                throw new IOException("Error while writing bloom filter to file system", e);
        }

        }
        }

        public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
        Configuration conf = new Configuration();

        Job job = Job.getInstance(conf, "BloomFilterCreation");
        job.setJarByClass(BFDriver.class);

        Path inputPath = new Path(args[0]);
        Path filterFolder = new Path(args[1]);
        String filterOutput = args[1] + Path.SEPARATOR + "filter";

        job.getConfiguration().set(BFReducer.FILTER_OUTPUT_FILE_CONF, filterOutput);

        job.setMapperClass(BFMapper.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setMapOutputKeyClass(NullWritable.class);
        job.setMapOutputValueClass(BloomFilter.class);

        job.setReducerClass(BFReducer.class);
        job.setOutputFormatClass(NullOutputFormat.class);
        job.setNumReduceTasks(1);

        FileInputFormat.setInputPaths(job, inputPath);
        FileOutputFormat.setOutputPath(job, filterFolder);

        System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}
```

b) Testing with stored filter

Driver with Mapper:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.bloom.BloomFilter;
import org.apache.hadoop.util.bloom.Key;

import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;

public class BTDriver {

	public static final String FILTER_FILE = "bloomfilter.filename";


	public static class BTMapper extends Mapper<LongWritable, Text, Text, Text>{
	private BloomFilter filter = new BloomFilter();

//	private static final Text TP_KEY = new Text("TRUE_POSITIVE");
//	private static final Text FP_KEY = new Text("FALSE_POSITIVE");
	private static final Text MP_KEY = new Text("MAYBE_POSITIVE");
	private static final Text FN_KEY = new Text("FALSE_NEGATIVE");

	private Text outval = new Text();

	@Override
	protected void setup(Context context) throws IOException, InterruptedException {
	String bfCacheFile = context.getConfiguration().get(FILTER_FILE);

	try(
		FileInputStream fis = new FileInputStream(bfCacheFile);
		DataInputStream dis = new DataInputStream(fis);
	){
		filter.readFields(dis);
	} catch (Exception e){
		throw new IOException("Error while reading bloom filter from file system", e);
	}

	}

	@Override
```

```
	protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
		String input = value.toString();
		String[] Tokens = input.split(",");

		if (!Tokens[0].equalsIgnoreCase("GAME_ID") && Tokens.length>=20) {
			String name = Tokens[5];
			boolean membership = filter.membershipTest(new Key(name.getBytes()));

			float pt3 = Float.parseFloat(Tokens[12]);
			float pt2 = Float.parseFloat(Tokens[9]) - Float.parseFloat(Tokens[12]);
			float pt1 = Float.parseFloat(Tokens[15]);

			float ptTotal = pt1*1 + pt2*2 + pt3*3;

			if (ptTotal>=40 && !membership){
			//False Negative
			outval.set(name);
			context.write(FN_KEY, outval);
			}
//			else if (ptTotal<40 && membership){
//			//False Positive
//			outval.set(name);
//			context.write(FP_KEY, outval);
//			}
			else if (membership){
			outval.set(name);
			context.write(MP_KEY, outval);
			}

		}
		}
		}

		public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
		Configuration conf = new Configuration();
		Job job = Job.getInstance(conf, "Test Bloom");
		job.setJarByClass(BTDriver.class);

		Path datasetPath = new Path(args[0]);
		Path bfPath = new Path(args[1]);
		Path outputPath = new Path(args[2]);
```

```
        job.addCacheFile(bfPath.toUri());
        job.getConfiguration().set(FILTER_FILE, bfPath.getName());

        job.setMapperClass(BTMapper.class);
        job.setInputFormatClass(TextInputFormat.class);
        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);

        job.setNumReduceTasks(0);

        FileInputFormat.setInputPaths(job, datasetPath);
        FileOutputFormat.setOutputPath(job, outputPath);

        System.exit(job.waitForCompletion(true) ? 0: 1);
        }
}
```

8) Intermediate analysis with Top 50 Filtering
Mapper:

```
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import sun.reflect.generics.tree.Tree;

import java.io.IOException;
import java.util.Map;
import java.util.TreeMap;

public class TFMapper extends Mapper<LongWritable, Text, NullWritable, Text> {
        private TreeMap<Float, String> tmap;

        @Override
        protected void setup(Context context) throws IOException, InterruptedException {
        tmap = new TreeMap<Float, String>();
        }

        @Override
        protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        String input = value.toString();
        String[] Tokens = input.split(",");
```

```
	if (!Tokens[0].equalsIgnoreCase("GAME_ID") && Tokens.length>=20 &&
!Tokens[12].isEmpty()){
	String name = Tokens[5];

	float pt3 = Float.parseFloat(Tokens[12]);
	float pt2 = Float.parseFloat(Tokens[9]) - Float.parseFloat(Tokens[12]);
	float pt1 = Float.parseFloat(Tokens[15]);

	float shootPct = pt1*1 + pt2*2 + pt3*3;
//	float shootPct = Float.parseFloat(Tokens[11]);

	tmap.put(shootPct, name);

	if (tmap.size() > 50){
		tmap.remove(tmap.firstKey());
	}
	}
	}

	@Override
	protected void cleanup(Context context) throws IOException, InterruptedException {
	for (Map.Entry<Float, String> entry: tmap.entrySet()){
	float pct = entry.getKey();
	String name = entry.getValue();
	String outval = name + "/" + (String.valueOf(pct));
	context.write(NullWritable.get(), new Text(outval));
	}
	}
}


Reducer:
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import sun.reflect.generics.tree.Tree;

import java.io.IOException;
import java.util.Map;
import java.util.TreeMap;

public class TFReducer extends Reducer<NullWritable, Text, Text, FloatWritable> {
```

```
	private TreeMap<Float, String> tmap2;

	@Override
	protected void setup(Context context) throws IOException, InterruptedException {
	tmap2 = new TreeMap<Float, String>();
	}

	@Override
	protected void reduce(NullWritable key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {

	for (Text val : values){
	String combined = val.toString();
	String[] combineArray = combined.split("/");

	String name = combineArray[0];
	float pct = Float.parseFloat(combineArray[1]);
	tmap2.put(pct, name);

	if (tmap2.size() > 50){
		tmap2.remove(tmap2.firstEntry());
	}
	}

	for (Map.Entry<Float, String> entry: tmap2.entrySet()){
	context.write(new Text(entry.getValue()), new FloatWritable(entry.getKey()));
	}
	}

//	@Override
//	protected void cleanup(Context context) throws IOException, InterruptedException {
//	for (Map.Entry<Float, String> entry: tmap2.entrySet()){
//	float pct = entry.getKey();
//	String name = entry.getValue();
//
//	context.write(new Text(name), new FloatWritable(pct));
//	}
//	}
}


Driver:
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
```

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class TFDriver {
	public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
	Configuration conf = new Configuration();
	FileSystem fs =FileSystem.get(conf);
	if(fs.exists(new Path(args[1])))
	{
	fs.delete(new Path(args[1]), true);
	}
	Job job = Job.getInstance(conf);
	job.setMapperClass(TFMapper.class);
	job.setReducerClass(TFReducer.class);
	job.setJarByClass(TFDriver.class);

	FileInputFormat.addInputPath(job, new Path(args[0]));
	FileOutputFormat.setOutputPath(job, new Path(args[1]));

	job.setInputFormatClass(TextInputFormat.class);
	job.setMapOutputKeyClass(NullWritable.class);
	job.setMapOutputValueClass(Text.class);
	job.setOutputKeyClass(Text.class);
	job.setOutputValueClass(FloatWritable.class);
//	job.setOutputValueClass(Text.class);
	System.exit(job.waitForCompletion(true) ? 0 : 1);
	}
}
```

9) Intermediate analysis with Inner Join

Games.csv Mapper:

```
import org.apache.hadoop.io.LongWritable;
```

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.Map;

public class GMapper extends Mapper<LongWritable, Text, Text, Text> {
        private Text outkey = new Text();
        private Text outval = new Text();

        @Override
        protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        String input = value.toString();
        String[] Tokens = input.split(",");

        String gameId = Tokens[1];
        String season = Tokens[5];

        if (!gameId.isEmpty() && !season.isEmpty()){
        outkey.set(gameId);
        outval.set("S"+season);
        context.write(outkey, outval);
        }
        }
}
```

Games_Details.csv Mapper:

```
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class GDMapper extends Mapper<LongWritable, Text, Text, Text> {
        private Text outkey = new Text();
        private Text outval = new Text();

        @Override
        protected void map(LongWritable key, Text value, Context context) throws IOException,
InterruptedException {
        String input = value.toString();
        String[] Tokens = input.split(",");
```

```
        String gameId = Tokens[0];
        String playerName = Tokens[5];

        if (!gameId.isEmpty() && !playerName.isEmpty() && Tokens.length>=27 &&
!Tokens[0].equalsIgnoreCase("GAME_ID")){
        float pt3 = Float.parseFloat(Tokens[12]);
        float pt2 = Float.parseFloat(Tokens[9]) - Float.parseFloat(Tokens[12]);
        float pt1 = Float.parseFloat(Tokens[15]);

        float totalPts = pt1*1 + pt2*2 + pt3*3;
        if (totalPts >= 40){
                outkey.set(gameId);
                outval.set("G"+playerName);
                context.write(outkey, outval);
        }

        }
        }
}
```

Join Reducer:

```
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;
import java.util.ArrayList;

public class JoinReducer extends Reducer<Text, Text, Text, Text> {

        private static final Text EMPTY_TEXT = new Text("");
        private Text tmp = new Text();
        private ArrayList<Text> seasonList = new ArrayList<Text>();
        private ArrayList<Text> playerList = new ArrayList<Text>();
        private String joinType = null;

        @Override
        protected void setup(Context context) throws IOException, InterruptedException {
        joinType = context.getConfiguration().get("join.type");
        }

        @Override
        protected void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
```

```
        seasonList.clear();
        playerList.clear();

        while (values.iterator().hasNext()){
        tmp = values.iterator().next();
        if (tmp.charAt(0) == 'S'){
                seasonList.add(new Text(tmp.toString().substring(1)));
        } else if (tmp.charAt(0)=='G'){
                playerList.add(new Text(tmp.toString().substring(1)));
        }
        }

        if (joinType.equalsIgnoreCase("inner")){
        if (!seasonList.isEmpty() && !playerList.isEmpty()){
                for (Text S: seasonList){
                for (Text G: playerList){
                context.write(S, G);
                }
                }
        }
        }
        }
}
```

Join Driver:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

import java.io.IOException;

public class JoinDriver {
        public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
        Configuration conf = new Configuration();
        FileSystem fs =FileSystem.get(conf);
```

```
        if(fs.exists(new Path(args[2])))
        {
        fs.delete(new Path(args[2]), true);
        }
        Job job = Job.getInstance(conf);

        MultipleInputs.addInputPath(job, new Path(args[0]), TextInputFormat.class,
GMapper.class);
        MultipleInputs.addInputPath(job, new Path(args[1]), TextInputFormat.class,
GDMapper.class);
        Path outputpath = new Path(args[2]);

        job.setReducerClass(JoinReducer.class);
        job.setJarByClass(JoinDriver.class);

        job.getConfiguration().set("join.type", "inner");
        FileOutputFormat.setOutputPath(job, outputpath);

        job.setOutputKeyClass(Text.class);
//      job.setOutputValueClass(FloatWritable.class);
        job.setOutputValueClass(Text.class);
        System.exit(job.waitForCompletion(true) ? 0 : 1);
        }
}
```

KeyValue InputFormat Mapper:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;

public class kvMapper extends Mapper<Text, Text, Text, Text> {
//      IntWritable result = new IntWritable();
        @Override
        protected void map(Text key, Text value, Context context) throws IOException,
InterruptedException {
//      int val = Integer.parseInt(value.toString());
//      result.set(val);
        context.write(key,value);
        }
```

```
}
```

KeyValue InputFormat Reducer:

```
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

public class kvReducer extends Reducer<Text, Text, Text, IntWritable> {
	IntWritable sum = new IntWritable(0);
	@Override
	protected void reduce(Text key, Iterable<Text> values, Context context) throws
IOException, InterruptedException {
	int count = 0;
	for (Text val : values){
	count++;
	}
	sum.set(count);
	context.write(key,sum);
	}
}
```

KeyValue InputFormat Driver:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.KeyValueTextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import java.io.IOException;

public class kvDriver {
	public static void main(String[] args) throws IOException, ClassNotFoundException,
InterruptedException {
```

```
        Configuration conf = new Configuration();
        FileSystem fs = FileSystem.get(conf);

        if(fs.exists(new Path(args[1]))){
        fs.delete(new Path(args[1]), true);
        }
        Job job = Job.getInstance(conf);
        job.setJarByClass(kvDriver.class);
        job.setMapperClass(kvMapper.class);
        job.setReducerClass(kvReducer.class);

        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        job.setInputFormatClass(KeyValueTextInputFormat.class);

        job.setMapOutputKeyClass(Text.class);
        job.setMapOutputValueClass(Text.class);

        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);

        System.exit(job.waitForCompletion(true)?0:1);
        }
}
```

10) Advanced Analysis - Converting CSV to Sequence File with Preprocessing

Custom Writable:

```
import org.apache.hadoop.io.FloatWritable;
import org.apache.hadoop.io.WritableComparable;

import java.io.DataInput;
import java.io.DataOutput;
import java.io.IOException;

public class GameDataWritable implements WritableComparable<GameDataWritable> {
        public String city;
        public float minutes;
        public String position;
        public float fgm;
        public float fga;
```

```
public float fg3m;
public float fg3a;
public float ftm;
public float fta;
public float oreb;
public float dreb;
public float ast;
public float stl;
public float blk;

@Override
public void readFields(DataInput dataInput) throws IOException {
city = dataInput.readUTF();
minutes = dataInput.readFloat();
position = dataInput.readUTF();
fgm = dataInput.readFloat();
fga = dataInput.readFloat();
fg3m = dataInput.readFloat();
fg3a = dataInput.readFloat();
ftm = dataInput.readFloat();
fta = dataInput.readFloat();
oreb = dataInput.readFloat();
dreb = dataInput.readFloat();
ast = dataInput.readFloat();
stl = dataInput.readFloat();
blk = dataInput.readFloat();
}

@Override
public void write(DataOutput dataOutput) throws IOException {
dataOutput.writeUTF(city);
dataOutput.writeFloat(minutes);
dataOutput.writeUTF(position);
dataOutput.writeFloat(fgm);
dataOutput.writeFloat(fga);
dataOutput.writeFloat(fg3m);
dataOutput.writeFloat(fg3a);
dataOutput.writeFloat(ftm);
dataOutput.writeFloat(fta);
dataOutput.writeFloat(oreb);
dataOutput.writeFloat(dreb);
dataOutput.writeFloat(ast);
dataOutput.writeFloat(stl);
dataOutput.writeFloat(blk);
```

```
}

@Override
public int compareTo(GameDataWritable o) {
return -1 * Float.compare(((this.fg3m*3)+((this.fgm-this.fg3m)*2)+this.ftm*1),
          ((o.fg3m*3)+((o.fgm-o.fg3m)*2)+o.ftm*1));
}

public String getCity() {
return city;
}

public void setCity(String city) {
this.city = city;
}

public float getMinutes() {
return minutes;
}

public void setMinutes(float minutes) {
this.minutes = minutes;
}

public String getPosition() {
return position;
}

public void setPosition(String position) {
this.position = position;
}

public float getFgm() {
return fgm;
}

public void setFgm(float fgm) {
this.fgm = fgm;
}

public float getFga() {
return fga;
}
```

```
public void setFga(float fga) {
this.fga = fga;
}

public float getFg3m() {
return fg3m;
}

public void setFg3m(float fg3m) {
this.fg3m = fg3m;
}

public float getFg3a() {
return fg3a;
}

public void setFg3a(float fg3a) {
this.fg3a = fg3a;
}

public float getFtm() {
return ftm;
}

public void setFtm(float ftm) {
this.ftm = ftm;
}

public float getFta() {
return fta;
}

public void setFta(float fta) {
this.fta = fta;
}

public float getOreb() {
return oreb;
}

public void setOreb(float oreb) {
this.oreb = oreb;
}
```

```
	public float getDreb() {
	return dreb;
	}

	public void setDreb(float dreb) {
	this.dreb = dreb;
	}

	public float getAst() {
	return ast;
	}

	public void setAst(float ast) {
	this.ast = ast;
	}

	public float getStl() {
	return stl;
	}

	public void setStl(float stl) {
	this.stl = stl;
	}

	public float getBlk() {
	return blk;
	}

	public void setBlk(float blk) {
	this.blk = blk;
	}
}
```

Driver:

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.compress.DefaultCodec;
import org.apache.hadoop.thirdparty.protobuf.compiler.PluginProtos;
```

```
import java.io.*;

public class CSVtoSeq {
	public static void main(String[] args) throws IOException {
	File inputFile = new File(args[0]);
	Path outputPath = new Path(args[1]);

	int processed =0;
	int errors =0;

	try(
		BufferedReader br = new BufferedReader(new FileReader(inputFile));

		SequenceFile.Writer writer = SequenceFile.createWriter(new Configuration(),
		SequenceFile.Writer.file(outputPath),
		SequenceFile.Writer.keyClass(NullWritable.class),
		SequenceFile.Writer.valueClass(GameDataWritable.class),
		SequenceFile.Writer.compression(SequenceFile.CompressionType.BLOCK, new
DefaultCodec()));
		){
	for (String line=br.readLine(); line!=null; line=br.readLine()){
		if (processed == 0){
		processed ++;
		continue;
		}

		try{
		String[] lineArray = line.split(",");
		if (lineArray.length>=27){
		GameDataWritable gd = new GameDataWritable();

		if (!lineArray[3].isEmpty() && !lineArray[8].isEmpty() &&
			!lineArray[6].isEmpty() && !lineArray[9].isEmpty() &&
			!lineArray[10].isEmpty() && !lineArray[12].isEmpty() &&
			!lineArray[13].isEmpty() && !lineArray[15].isEmpty() &&
			!lineArray[16].isEmpty() && !lineArray[18].isEmpty() &&
			!lineArray[19].isEmpty() && !lineArray[21].isEmpty() &&
			!lineArray[22].isEmpty() && !lineArray[23].isEmpty()){
			gd.setCity(lineArray[3]);
			String[] time = lineArray[8].split(":");
			gd.setMinutes(Float.parseFloat(time[0]));
			gd.setPosition(lineArray[6]);
			gd.setFgm(Float.parseFloat(lineArray[9]));
			gd.setFga(Float.parseFloat(lineArray[10]));
```

```
                        gd.setFg3m(Float.parseFloat(lineArray[12]));
                        gd.setFg3a(Float.parseFloat(lineArray[13]));
                        gd.setFtm(Float.parseFloat(lineArray[15]));
                        gd.setFta(Float.parseFloat(lineArray[16]));
                        gd.setOreb(Float.parseFloat(lineArray[18]));
                        gd.setDreb(Float.parseFloat(lineArray[19]));
                        gd.setAst(Float.parseFloat(lineArray[21]));
                        gd.setBlk(Float.parseFloat(lineArray[22]));
                        gd.setStl(Float.parseFloat(lineArray[23]));

                        writer.append(NullWritable.get(), gd);
                }
                }

                } catch(Exception e){
                errors++;
                }
                processed++;
                if (processed % 100000 ==0){
                System.out.println(String.format("%d hundred thousand lines processed",
processed/100000));
                }
        }
        } finally {
        System.out.println("Number of lines processed = " + processed);
        System.out.println("Number of Errors = "+ errors);
        }
        }
}
```

11) Advanced analysis with Python:
In[5]:

```
import pandas as pd
import numpy as np
from math import pi

import matplotlib.pyplot as plt
import seaborn as sns

from IPython.display import display, Markdown
```

```
# In[8]:


games_details = pd.read_csv('C:/Users/Karti/NEU/data/NBA/games_details.csv')
players = pd.read_csv('C:/Users/Karti/NEU/data/NBA/players.csv')
teams = pd.read_csv('C:/Users/Karti/NEU/data/NBA/teams.csv')
ranking = pd.read_csv('C:/Users/Karti/NEU/data/NBA/ranking.csv')
games  = pd.read_csv('C:/Users/Karti/NEU/data/NBA/games.csv')


# In[10]:


games_details.isnull().sum() #Missing values per column


# In[11]:


# Who are the players with most games played ?
players_name = games_details['PLAYER_NAME']
val_cnt = players_name.value_counts().to_frame().reset_index()
val_cnt.columns = ['PLAYER_NAME', 'Number of games']


# In[13]:


def plot_top(df, column, label_col=None, max_plot=5):
        top_df = df.sort_values(column, ascending=False).head(max_plot)

        height = top_df[column]
        x = top_df.index if label_col == None else top_df[label_col]

        gold, silver, bronze, other = ('#FFA400', '#bdc3c7', '#cd7f32', '#3498db')
        colors = [gold if i == 0 else silver if i == 1 else bronze if i == 2 else other for i in range(0,
len(top_df))]

        fig, ax = plt.subplots(figsize=(18, 7))
        ax.bar(x, height, color=colors)
        plt.xticks(x, x, rotation=60)
        plt.xlabel(label_col)
        plt.ylabel(column)
```

```
        plt.title(f'Top {max_plot} of {column}')
        plt.show()


# In[14]:


plot_top(val_cnt, column='Number of games', label_col='PLAYER_NAME', max_plot=10)


# In[15]:


def convert_min(x):
        if pd.isna(x):
        return 0
        x = str(x).split(':')
        if len(x) < 2:
        return int(x[0])
        else:
        return int(x[0])*60+int(x[1])


# In[16]:


df_tmp = games_details[['PLAYER_NAME', 'MIN']]
df_tmp.loc[:,'MIN'] = df_tmp['MIN'].apply(convert_min)
agg = df_tmp.groupby('PLAYER_NAME').agg('sum').reset_index()
agg.columns = ['PLAYER_NAME', 'Number of seconds played']
plot_top(agg, column='Number of seconds played', label_col='PLAYER_NAME', max_plot=10)


# In[17]:


games_details.columns


# In[35]:


#MACHINE LEARNING WITH SCIKIT LEARN
```

```
#Remove unnecessary columsn
#Handle/Remove missing values

gd = games_details.drop(columns = ['GAME_ID', 'TEAM_ID','TEAM_CITY',
'TEAM_ABBREVIATION', 'PLAYER_ID', 'PLAYER_NAME','COMMENT', 'MIN', 'PTS',
'PLUS_MINUS'])

gd.head()


# In[36]:


gd.isnull().sum()


# In[37]:


gd = gd.dropna()


# In[38]:


gd.isnull().sum()


# In[39]:


gd.describe()


# In[40]:


gd['START_POSITION'].unique()


# In[41]:


y = gd['START_POSITION']
```

```
X = gd.drop(columns=['START_POSITION'])

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)


# In[42]:


from sklearn.neighbors import KNeighborsClassifier
neigh = KNeighborsClassifier(n_neighbors=3)
neigh.fit(X_train, y_train)


# In[43]:


neigh.predict(X_test)


# In[44]:


neigh.score(X_test, y_test)


# In[45]:


from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# In[46]:


model = XGBClassifier()
model.fit(X_train, y_train)


# In[48]:
```

```
y_pred = model.predict(X_test)
y_pred


# In[50]:


accuracy = accuracy_score(y_test, y_pred)
accuracy


# In[52]:


model1 = XGBClassifier(base_score=1, learning_rate=0.100000012)
model1.fit(X_train, y_train)

y_pred1 = model1.predict(X_test)

accuracy = accuracy_score(y_test, y_pred1)
print(accuracy)
```