BIG DATA PROGRAMMING

Coursework

Yuwarajalingam Dinesh

20221733

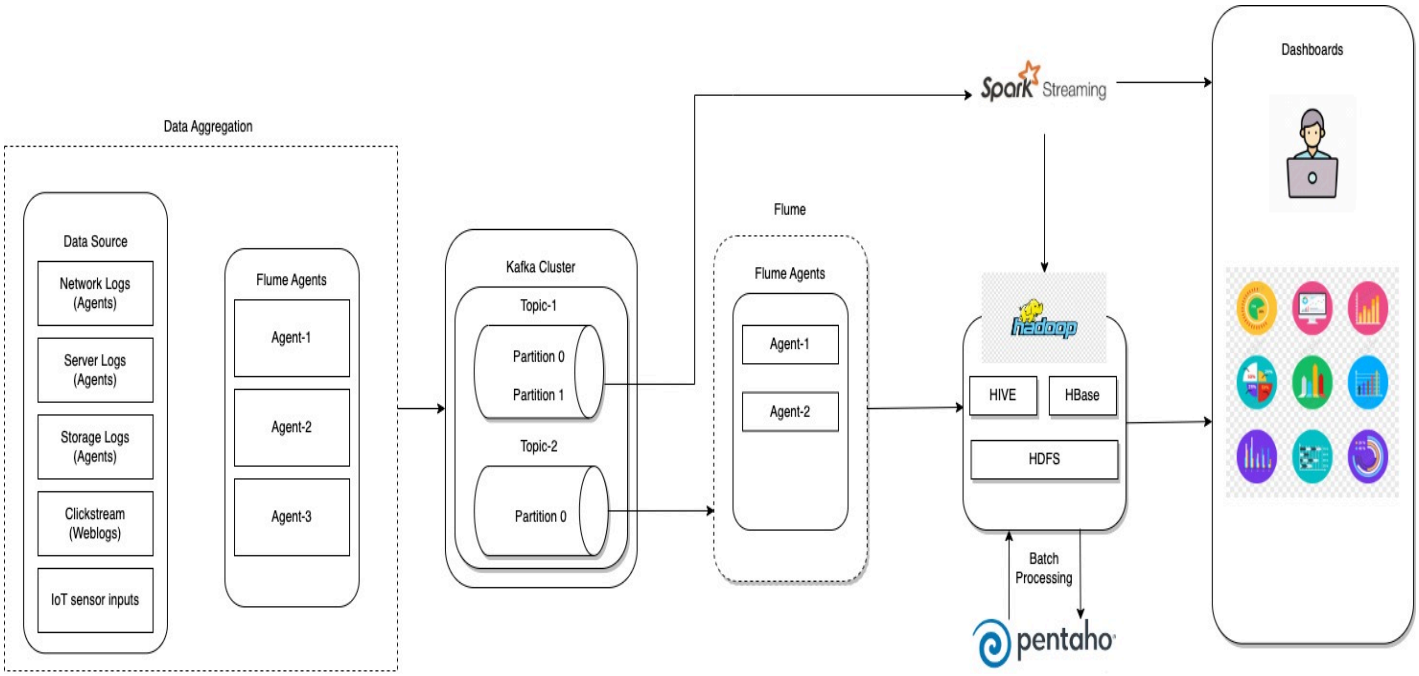
2237630

2022 September Batch

Table of Contents

Table of figures

1. System Architectural Design



Flume agents are used to collect the data from the Data sources and passing them to kafka cluster to store them in the que. when the flume consumers are ready data will be consumed and will be pushed to HDFS to store the data for the historical data processing. In the meantime spark streaming will also consume these data and based on the end users requirements data will be modeled. this is based on the real time analytics. for the historical data, pentaho tool is used to process the data periodically and from the Hadoop, endusers can visualize the data.

Below comes the description of the each component, its duties and the interaction with each other.

Flume agent on servers:

Flume is a lightweight java program which runs on JVM and collects log messages from servers. This agent is configured to read the log file and push the log messages ('event' as per Flume's language) into the Data collector i.e. Kafka broker on configured batch size or time interval.

Data Collector/Kafka Broker:

Apache Kafka is chosen as the data collector because of its scalability, durability and low-latency advantages. This module collects all the log messages pushed by the flume agents and keeps it safe for consumers to consume.

Cluster and Load Balancing of Kafka broker:

For high availability and durability of the messages, I choose to have multi node-multi broker setup. In this diagram, all the log messages are sent to a single topic T and all the flume agents will logically push them to that topic.

Flume Consumer:

This layer consists of group of flume consumer agents connected to Kafka broker and subscribed to topic T. These agents together, will read the data from the subscribed topic and syncs into an intermediate storage area which will be of HDFS files.

Load balancing on consumer side:

Each of these agents will decide among themselves who will read from which partition and do load-balancing by themselves. These agents will be configured to sync the consumed log messages into HDFS files. To achieve this, all these agents are being grouped into one consumer group, so that they can work in parallel.

Spark Streaming:

Spark Streaming is an extension of the core Spark API that allows data engineers and data scientists to process real-time data from various sources including Kafka, Flume, and Amazon Kinesis. This processed data can be pushed out to file systems, databases, and live dashboards.

Pentaho:

Pentaho is an extensively used Business Intelligence tool set (suite) across industries for data management. Analysts, data managers, software developers, and even students find the applicability of this tool. Pentaho suite enhances the overall performance of the business by generating informative reports in varied formats like text, XML, HTML, CSV, Excel, PDF, etc.

2. Data Analysis

2.1 MapReduce

The code is given in the attached folder itself. Please refer for map reduce part.

1.

```
22/12/22 06:35:18 INFO mapreduce.Job: Counters: 33
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=115444
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=20719246
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=5
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=27475
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=27475
                Total vcore-seconds taken by all map tasks=27475
                Total megabyte-seconds taken by all map tasks=28134400
        Map-Reduce Framework
                Map input records=193469
                Map output records=0
                Input split bytes=112
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=3334
                CPU time spent (ms)=11290
                Physical memory (bytes) snapshot=169058304
                Virtual memory (bytes) snapshot=736563200
                Total committed heap usage (bytes)=146276352
        BallResult
                NO_OF_DELIVERIES_EXTRA_RUNS_GIVEN=10233
                NO_OF_DELIVERIES_NO_RUNS_GIVEN=67841
                NO_OF_DELIVERIES_WICKET_WAS_TAKEN=9495
        File Input Format Counters
                Bytes Read=20719134
        File Output Format Counters
                Bytes Written=0
NO_OF_DELIVERIES_EXTRA_RUNS_GIVEN       10233
NO_OF_DELIVERIES_NO_RUNS_GIVEN  67841
NO_OF_DELIVERIES_WICKET_WAS_TAKEN       9495
```

Figure 1:Map Reduce log for first question

```
                Total committed heap usage (bytes)=146276352
        BallResult
                NO_OF_DELIVERIES_EXTRA_RUNS_GIVEN=10233
                NO_OF_DELIVERIES_NO_RUNS_GIVEN=67841
                NO_OF_DELIVERIES_WICKET_WAS_TAKEN=9495
```

Figure 2:Answer Calculated for MapReduce first question

```java
public void map(Object key, Text value,Context context) throws IOException, InterruptedException {
    Map<String, String> parsed = MRDPUtils.getMapFromCSV(value.toString());
    int is_wicket=-1;
    int extra_runs=-1;
    int total_runs=-1;

    if(parsed.get("is_wicket")!=null && !parsed.get("is_wicket").isEmpty()){
        try{
            is_wicket = Integer.parseInt((String)parsed.get("is_wicket"));
        }catch (Exception e){
            System.err.println(e);
        }
    }

    if((String)parsed.get("extra_runs")!=null && !((String) parsed.get("extra_runs")).isEmpty()){
        try{
            extra_runs = Integer.parseInt((String)parsed.get("extra_runs"));
        }catch (Exception e){
            System.err.println(e);
        }
    }

    if((String)parsed.get("total_runs")!=null && !((String) parsed.get("total_runs")).isEmpty()){
        try{
            total_runs = Integer.parseInt((String)parsed.get("total_runs"));
        }catch (Exception e){
            System.err.println(e);
        }
    }
    if(is_wicket==1){
        context.getCounter( s: "BallResult",NO_OF_DELIVERIES_WICKET_WAS_TAKEN).increment( l: 1);
    }

    if(extra_runs > 0){
        context.getCounter( s: "BallResult",NO_OF_DELIVERIES_EXTRA_RUNS_GIVEN).increment( l: 1);
    }
    if (total_runs==0){
        context.getCounter( s: "BallResult",NO_OF_DELIVERIES_NO_RUNS_GIVEN).increment( l: 1);
    }
}
```

Figure 3:Code for MapReduce1

2.

```
22/12/22 06:39:15 INFO mapreduce.Job: Counters: 45
        File System Counters
                FILE: Number of bytes read=0
                FILE: Number of bytes written=115439
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=20719246
                HDFS: Number of bytes written=0
                HDFS: Number of read operations=5
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=4171
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=4171
                Total vcore-seconds taken by all map tasks=4171
                Total megabyte-seconds taken by all map tasks=4271104
        Map-Reduce Framework
                Map input records=193469
                Map output records=0
                Input split bytes=112
                Spilled Records=0
                Failed Shuffles=0
                Merged Map outputs=0
                GC time elapsed (ms)=52
                CPU time spent (ms)=1370
                Physical memory (bytes) snapshot=169967616
                Virtual memory (bytes) snapshot=730394624
                Total committed heap usage (bytes)=146276352
        WicketsByTeam
                Chennai Super Kings=1104
                Deccan Chargers=446
                Delhi Capitals=212
                Delhi Daredevils=912
                Gujarat Lions=149
                Kings XI Punjab=1070
                Kochi Tuskers Kerala=74
                Kolkata Knight Riders=1079
                Mumbai Indians=1237
                NA=10
                Pune Warriors=235
                Rajasthan Royals=910
                Rising Pune Supergiants=189
                Royal Challengers Bangalore=1115
                Sunrisers Hyderabad=753
        File Input Format Counters
                Bytes Read=20719134
        File Output Format Counters
                Bytes Written=0
Chennai Super Kings     1104
Deccan Chargers 446
Delhi Capitals  212
Delhi Daredevils        912
Gujarat Lions   149
Kings XI Punjab 1070
Kochi Tuskers Kerala    74
Kolkata Knight Riders   1079
Mumbai Indians  1237
NA      10
Pune Warriors   235
Rajasthan Royals        910
Rising Pune Supergiants 189
Royal Challengers Bangalore     1115
Sunrisers Hyderabad     753
```

Figure 4:MapReduce Log for second question

```
        Total committed heap usage (bytes)=14827
WicketsByTeam
        Chennai Super Kings=1104
        Deccan Chargers=446
        Delhi Capitals=212
        Delhi Daredevils=912
        Gujarat Lions=149
        Kings XI Punjab=1070
        Kochi Tuskers Kerala=74
        Kolkata Knight Riders=1079
        Mumbai Indians=1237
        NA=10
        Pune Warriors=235
        Rajasthan Royals=910
        Rising Pune Supergiants=189
```

Figure 5:Answer for MapReduce second question

```java
public static class WicketsByTeam extends Mapper<Object, Text, NullWritable, NullWritable> {

    public void map(Object key, Text value,Context context) throws IOException, InterruptedException {
        Map<String, String> parsed = MRDPUtils.getMapFromCSV(value.toString());
        int is_wicket=0;
        String team =parsed.get("bowling_team");

        if(parsed.get("is_wicket")!=null && !parsed.get("is_wicket").isEmpty()){
            try{
                is_wicket = Integer.parseInt((String)parsed.get("is_wicket"));
            }catch (Exception e){
                System.err.println(e);
            }
        }

        if(is_wicket==1){
            context.getCounter( s: "WicketsByTeam",team).increment( l: 1);
        }
    }
}
```

Figure 6:Code for MapReduce2

2.2 Hive

1.

```
hive> select batting_team,sum(total_runs)total_runs from ipl_dataset
    > GROUP BY
    > batting_team
    > ORDER BY total_runs DESC
    > LIMIT 10;
Query ID = root_20221222071600_67c3d1a1-0a40-4247-ac09-d5d810506445
Total jobs = 2
Launching Job 1 out of 2
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_1671337528276_0019, Tracking URL = http://57d8be319d11:8088/proxy/application_1671337528276_0019/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1671337528276_0019
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-12-22 07:16:30,430 Stage-1 map = 0%,  reduce = 0%
2022-12-22 07:17:07,732 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 21.11 sec
2022-12-22 07:17:22,001 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 23.78 sec
MapReduce Total cumulative CPU time: 23 seconds 780 msec
Ended Job = job_1671337528276_0019
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 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_1671337528276_0020, Tracking URL = http://57d8be319d11:8088/proxy/application_1671337528276_0020/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1671337528276_0020
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2022-12-22 07:17:44,028 Stage-2 map = 0%,  reduce = 0%
2022-12-22 07:17:49,510 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 0.83 sec
2022-12-22 07:17:57,784 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.29 sec
MapReduce Total cumulative CPU time: 2 seconds 290 msec
Ended Job = job_1671337528276_0020
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 23.78 sec   HDFS Read: 20727806 HDFS Write: 635 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.29 sec   HDFS Read: 5338 HDFS Write: 244 SUCCESS
Total MapReduce CPU Time Spent: 26 seconds 70 msec
OK
Mumbai Indians  32286
Royal Challengers Bangalore     30214
Kings XI Punjab 30017
Kolkata Knight Riders   29383
Chennai Super Kings     28363
Rajasthan Royals        24507
Delhi Daredevils        24285
Sunrisers Hyderabad     19332
Deccan Chargers 11463
Pune Warriors   6358
Time taken: 118.851 seconds, Fetched: 10 row(s)
```

Figure 7:Hive Logs for question 1

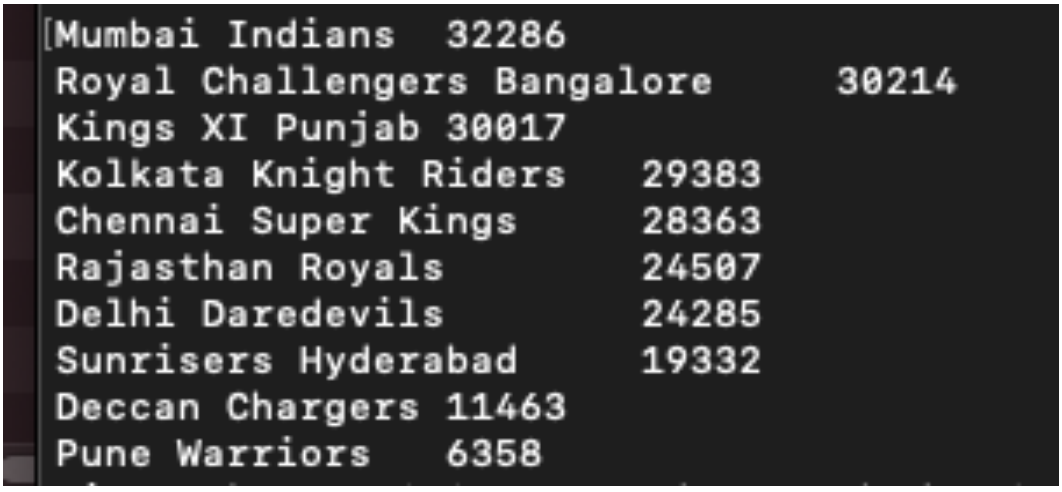


Figure 8:Answer for hive question 1

```
hive> select batting_team,sum(total_runs)total_runs from ipl_dataset
    > GROUP BY
    > batting_team
    > ORDER BY total_runs DESC
    > LIMIT 10;
```

Figure 9:hive query for first question

2.

```
hive> select `over`,AVG(total_runs) AS run_rate
    > from ipl_dataset
    > group by `over`
    > order by `over`;
Query ID = root_20221222074027_04296585-3454-492b-8638-30ac29b692a1
Total jobs = 2
Launching Job 1 out of 2
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_1671337528276_0021, Tracking URL = http://57d8be319d11:8088/proxy/application_1671337528276_0021/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1671337528276_0021
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-12-22 07:40:44,468 Stage-1 map = 0%,  reduce = 0%
2022-12-22 07:40:53,402 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 3.21 sec
2022-12-22 07:41:00,042 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.43 sec
MapReduce Total cumulative CPU time: 4 seconds 430 msec
Ended Job = job_1671337528276_0021
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 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_1671337528276_0022, Tracking URL = http://57d8be319d11:8088/proxy/application_1671337528276_0022/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1671337528276_0022
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2022-12-22 07:41:19,528 Stage-2 map = 0%,  reduce = 0%
2022-12-22 07:41:29,246 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.45 sec
2022-12-22 07:41:36,038 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.66 sec
MapReduce Total cumulative CPU time: 2 seconds 660 msec
Ended Job = job_1671337528276_0022
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.43 sec   HDFS Read: 20728332 HDFS Write: 616 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.66 sec   HDFS Read: 5150 HDFS Write: 427 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 90 msec
OK
0       0.9561146869514335
1       1.1352510970258411
2       1.2806499261447564
3       1.3196243203163618
4       1.3282798256044392
5       1.3380351643985298
6       1.0817283704221137
7       1.155608926248374
8       1.2042352468887996
9       1.1970766129032258
10      1.236719383617194
11      1.2669582019729482
12      1.2790128492759536
13      1.3279448105436573
14      1.3826520024901432
15      1.4191199746755303
16      1.4948706988672793
17      1.574620294959278
18      1.6180986988629704
19      1.7588371775624403
Time taken: 71.294 seconds, Fetched: 20 row(s)
```

Figure 10:Hive Logs for second question

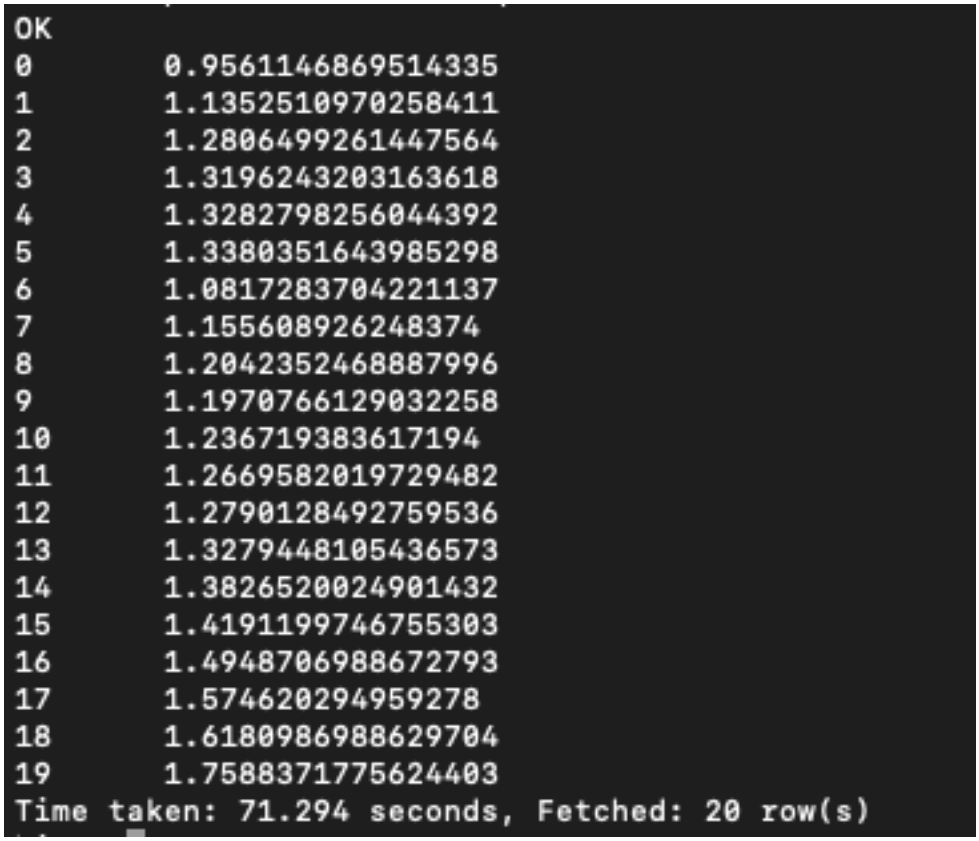


Figure 11:Answer for hive second question

```
hive> select `over`,AVG(total_runs) AS run_rate
    > from ipl_dataset
    > group by `over`
    > order by `over`;
```

Figure 12:Query for hive second question

2.3 Spark

1.

```
%sql
SELECT (SELECT COUNT(DISTINCT(batsman)) FROM
(SELECT DISTINCT(batsman),SUM(batsman_runs) AS total_runs,inning FROM ipl
GROUP BY batsman,inning,id)
WHERE total_runs>=50)*100/(SELECT COUNT(DISTINCT(batsman)) from ipl)
```

settings

(CAST((scalarsubquery() * CAST(100 AS BIGINT)) AS DOUBLE) / CAST(scalarsubquery() AS DOUBLE))
29.050279329608937

Figure 13:Spark first question query and answer

2.

```
%sql
SELECT f.first_team, f.first_innings_score,s.second_team,s.second_innings_score
FROM (SELECT batting_team As first_team,score AS first_innings_score,inning,id from
(SELECT * FROM
(SELECT DISTINCT(id),batting_team, SUM(total_runs) AS score,inning FROM ipl
GROUP BY batting_team,inning,id)
ORDER BY id,inning)
WHERE inning=1) f, (SELECT batting_team As second_team,score AS second_innings_score,inning,id from
(SELECT * FROM
(SELECT DISTINCT(id),batting_team, SUM(total_runs) AS score,inning FROM ipl
GROUP BY batting_team,inning,id)
ORDER BY id,inning)
WHERE inning=2) s
WHERE f.id=s.id
```

Figure 15:SQL query to transform data to calculate the result

first_team	first_innings_score	second_team	second_innings_score
Sunrisers Hyderabad	207.0	Royal Challengers Bangalore	172.0
Mumbai Indians	184.0	Rising Pune Supergiants	187.0
Gujarat Lions	183.0	Kolkata Knight Riders	184.0
Rising Pune Supergiants	163.0	Kings XI Punjab	164.0
Royal Challengers Bangalore	157.0	Delhi Daredevils	142.0
Gujarat Lions	135.0	Sunrisers Hyderabad	140.0
Kolkata Knight Riders	178.0	Mumbai Indians	180.0
Royal Challengers Bangalore	148.0	Kings XI Punjab	150.0

Figure 14:Result of the transformed data

```
%pyspark
from pyspark.sql.functions import when,countDistinct,col
df4 = sqlDF.withColumn("home", when(col("first_innings_score") > col("second_innings_score"),"won")
                               .when(col("first_innings_score") < col("second_innings_score"),"lost")
                               .otherwise("Drawn"))

df5 = sqlDF.withColumn("away", when(col("second_innings_score") < col("first_innings_score"),"lost")
                               .when(col("second_innings_score") > col("first_innings_score"),"won")
                               .otherwise("Drawn"))

df6 = df4.select("first_team","home")
df7 = df5.select("second_team","away")

Home = df6.withColumnRenamed("first_team","team") \
    .withColumnRenamed("home","outcome")

Away = df7.withColumnRenamed("second_team","team") \
    .withColumnRenamed("away","outcome")

data_new=Home.union(Away)

from pyspark.sql.functions import count
pivotDF = data_new.groupBy("team") \
  .pivot("outcome",['won','lost','Drawn']) \
  .agg(count("outcome"))

pivotDF.show()
```

```
+--------------------+---+----+-----+
|                team|won|lost|Drawn|
+--------------------+---+----+-----+
| Sunrisers Hyderabad| 67|  54|    3|
| Chennai Super Kings|106|  71|    1|
|     Deccan Chargers| 29|  46| null|
|Kochi Tuskers Kerala|  6|   8| null|
|    Rajasthan Royals| 79|  78|    3|
|       Gujarat Lions| 13|  16|    1|
|Royal Challengers...| 89| 102|    3|
|Kolkata Knight Ri...| 96|  92|    4|
|Rising Pune Super...| 14|  16| null|
|     Kings XI Punjab| 85| 101|    4|
|       Pune Warriors| 12|  33| null|
|    Delhi Daredevils| 70|  89|    1|
|      Delhi Capitals| 17|  14|    2|
|      Mumbai Indians|118|  81|    4|
+--------------------+---+----+-----+
```

Figure 16:Answer for the spark second question

3. Spark MLlib

Import Spark MLlib session from spark

```
%pyspark
from pyspark.sql import SparkSession
spark = SparkSession \
    .builder \
    .appName("Spark_MLlib") \
    .getOrCreate()
```

Figure 17:Create spark MLlib session

Importing data and show to confirm data has imported

```
%pyspark

df = spark.read.option("header",True).csv("/data/ipl-data.csv")
df.show()
```

Figure 19:Importing Data code

```
+------+------+----+----+-----------+-----------+---------+-----------+----------+----------+-
|    id|inning|over|ball|    batsman|non_striker|   bowler|batsman_runs|extra_runs|total_runs|n
+------+------+----+----+-----------+-----------+---------+-----------+----------+----------+-
|335982|     1|   6|   5| RT Ponting|BB McCullum|AA Noffke|          1|         0|         1|
|335982|     1|   6|   6|BB McCullum| RT Ponting|AA Noffke|          1|         0|         1|
|335982|     1|   7|   1|BB McCullum| RT Ponting|   Z Khan|          0|         0|         0|
|335982|     1|   7|   2|BB McCullum| RT Ponting|   Z Khan|          1|         0|         1|
|335982|     1|   7|   3| RT Ponting|BB McCullum|   Z Khan|          1|         0|         1|
|335982|     1|   7|   4|BB McCullum| RT Ponting|   Z Khan|          1|         0|         1|
|335982|     1|   7|   5| RT Ponting|BB McCullum|   Z Khan|          1|         0|         1|
|335982|     1|   7|   6|BB McCullum| RT Ponting|   Z Khan|          1|         0|         1|
|335982|     1|   8|   1|BB McCullum| RT Ponting|JH Kallis|          0|         0|         0|
|335982|     1|   8|   2|BB McCullum| RT Ponting|JH Kallis|          0|         0|         0|
|335982|     1|   8|   3|BB McCullum| RT Ponting|JH Kallis|          0|         0|         0|
|335982|     1|   8|   4|BB McCullum| RT Ponting|JH Kallis|          1|         0|         1|
|335982|     1|   8|   5| RT Ponting|BB McCullum|JH Kallis|          1|         0|         1|
|335982|     1|   8|   6|BB McCullum| RT Ponting|JH Kallis|          2|         0|         2|
|335982|     1|   9|   1| RT Ponting|BB McCullum| SR Joshi|          1|         0|         1|
```

Figure 18:Data imported screen shot

Create temporary view to execute sql queries

```
%pyspark
df.createOrReplaceTempView("ipl")
sqlDF = spark.sql("SELECT * FROM ipl")
sqlDF.show()
```

Figure 20:Creating temporary view ipl

```
+------+------+----+----+-----------+-----------+---------+------------+----------+----------+------------+---------+-
|    id|inning|over|ball|    batsman|non_striker|  bowler|batsman_runs|extra_runs|total_runs|non_boundary|is_wicket|d
+------+------+----+----+-----------+-----------+---------+------------+----------+----------+------------+---------+-
|335982|     1|   6|   5| RT Ponting|BB McCullum|AA Noffke|           1|         0|         1|           0|        0|
|335982|     1|   6|   6|BB McCullum| RT Ponting|AA Noffke|           1|         0|         1|           0|        0|
|335982|     1|   7|   1|BB McCullum| RT Ponting|  Z Khan|           0|         0|         0|           0|        0|
|335982|     1|   7|   2|BB McCullum| RT Ponting|  Z Khan|           1|         0|         1|           0|        0|
|335982|     1|   7|   3| RT Ponting|BB McCullum|  Z Khan|           1|         0|         1|           0|        0|
|335982|     1|   7|   4|BB McCullum| RT Ponting|  Z Khan|           1|         0|         1|           0|        0|
|335982|     1|   7|   5| RT Ponting|BB McCullum|  Z Khan|           1|         0|         1|           0|        0|
|335982|     1|   7|   6|BB McCullum| RT Ponting|  Z Khan|           1|         0|         1|           0|        0|
|335982|     1|   8|   1|BB McCullum| RT Ponting|JH Kallis|           0|         0|         0|           0|        0|
|335982|     1|   8|   2|BB McCullum| RT Ponting|JH Kallis|           0|         0|         0|           0|        0|
|335982|     1|   8|   3|BB McCullum| RT Ponting|JH Kallis|           0|         0|         0|           0|        0|
|335982|     1|   8|   4|BB McCullum| RT Ponting|JH Kallis|           1|         0|         1|           0|        0|
|335982|     1|   8|   5| RT Ponting|BB McCullum|JH Kallis|           1|         0|         1|           0|        0|
|335982|     1|   8|   6|BB McCullum| RT Ponting|JH Kallis|           2|         0|         2|           0|        0|
|335982|     1|   9|   1| RT Ponting|BB McCullum| SR Joshi|           1|         0|         1|           0|        0|
```

Figure 21:Temporary view selected data

Sql query to get the powerplay runs of each team against their relevant opponent team

```
%sql
SELECT batting_team,bowling_team,SUM(total_runs)
FROM ipl
WHERE over<6
GROUP BY batting_team,bowling_team,id
```

Figure 22:Sql query to get the power play score of each team

batting_team	bowling_team	sum(CAST(total_runs AS DOUBLE))
Rajasthan Royals	Kings XI Punjab	38.0
Delhi Daredevils	Mumbai Indians	60.0
Rajasthan Royals	Kolkata Knight Riders	36.0
Royal Challengers Bangalore	Chennai Super Kings	33.0
Kolkata Knight Riders	Delhi Daredevils	31.0
Kings XI Punjab	Mumbai Indians	46.0
Chennai Super Kings	Delhi Capitals	58.0
Chennai Super Kings	Mumbai Indians	40.0

Figure 23:Power play each team against their opponent

Making sql data frame using the above sql query

```
%pyspark
from pyspark.sql.functions import countDistinct,col
df.createOrReplaceTempView("ipl_powerplay_summary")
sqlDF1 = spark.sql("SELECT batting_team,bowling_team,SUM(total_runs) AS Power_play_score FROM ipl WHERE over<6 GROUP BY batting_team,bowling_team,id")
sqlDF1.show()
```

Figure 24:Making data frame using relevant sql query

Casting power play score to float

```
%pyspark
from pyspark.sql.functions import col
dataset = sqlDF1.select(col('batting_team'),
                        col('bowling_team'),
                        col('Power_play_score').cast('float')
                       )
dataset.show()
```

```
+--------------------+--------------------+---------------+
|        batting_team|        bowling_team|Power_play_score|
+--------------------+--------------------+---------------+
|    Rajasthan Royals|     Kings XI Punjab|           38.0|
|    Delhi Daredevils|      Mumbai Indians|           60.0|
|    Rajasthan Royals|Kolkata Knight Ri...|           36.0|
|Royal Challengers...| Chennai Super Kings|           33.0|
|Kolkata Knight Ri...|    Delhi Daredevils|           31.0|
|     Kings XI Punjab|      Mumbai Indians|           46.0|
| Chennai Super Kings|      Delhi Capitals|           58.0|
| Chennai Super Kings|      Mumbai Indians|           40.0|
|    Rajasthan Royals| Chennai Super Kings|           44.0|
|      Mumbai Indians|Kolkata Knight Ri...|           40.0|
|Kolkata Knight Ri...|    Delhi Daredevils|           54.0|
| Sunrisers Hyderabad|     Kings XI Punjab|           33.0|
|Kolkata Knight Ri...|Royal Challengers...|           53.0|
| Chennai Super Kings|      Mumbai Indians|           90.0|
|Kolkata Knight Ri...|    Delhi Daredevils|           47.0|
```

Figure 25:Casting powerplay score to float

Selecting data where when all the columns in dataset isnull

```
%pyspark
from pyspark.sql.functions import isnull, when, count, col
dataset.select([count(when(isnull(c), c)).alias(c) for c in dataset.columns]).show()
```

```
+------------+------------+----------------+
|batting_team|bowling_team|Power_play_score|
+------------+------------+----------------+
|           0|           0|               0|
+------------+------------+----------------+
```

Figure 26:Selecting data where when all the columns in dataset isnull

Mapping batting_team and bowling_team to numbers where machine learning model can understand as input

```
%pyspark
from pyspark.ml.feature import StringIndexer
dataset = StringIndexer(
    inputCol='batting_team',
    outputCol='ba_team',
    handleInvalid='keep').fit(dataset).transform(dataset)
dataset = StringIndexer(
    inputCol='bowling_team',
    outputCol='bo_team',
    handleInvalid='keep').fit(dataset).transform(dataset)
dataset.show()
```

Figure 27:Mapping string values into numbers

batting_team	bowling_team	Power_play_score	ba_team	bo_team
Rajasthan Royals	Kings XI Punjab	38.0	6.0	3.0
Delhi Daredevils	Mumbai Indians	60.0	5.0	0.0
Rajasthan Royals	Kolkata Knight Ri...	36.0	6.0	2.0
Royal Challengers...	Chennai Super Kings	33.0	1.0	4.0
Kolkata Knight Ri...	Delhi Daredevils	31.0	2.0	6.0
Kings XI Punjab	Mumbai Indians	46.0	3.0	0.0
Chennai Super Kings	Delhi Capitals	58.0	4.0	10.0
Chennai Super Kings	Mumbai Indians	40.0	4.0	0.0
Rajasthan Royals	Chennai Super Kings	44.0	6.0	4.0
Mumbai Indians	Kolkata Knight Ri...	40.0	0.0	2.0
Kolkata Knight Ri...	Delhi Daredevils	54.0	2.0	6.0
Sunrisers Hyderabad	Kings XI Punjab	33.0	7.0	3.0
Kolkata Knight Ri...	Royal Challengers...	53.0	2.0	1.0
Chennai Super Kings	Mumbai Indians	90.0	4.0	0.0
Kolkata Knight Ri...	Delhi Daredevils	47.0	2.0	6.0

Figure 28:transformed dataset

Assemble all the features with vector assembler

```
%pyspark
# Assemble all the features with VectorAssembler
required_features = ['ba_team','bo_team']
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=required_features, outputCol='features')
transformed_data = assembler.transform(dataset)
transformed_data.show()
```

batting_team	bowling_team	Power_play_score	ba_team	bo_team	features
Rajasthan Royals	Kings XI Punjab	38.0	6.0	3.0	[6.0,3.0]
Delhi Daredevils	Mumbai Indians	60.0	5.0	0.0	[5.0,0.0]
Rajasthan Royals	Kolkata Knight Ri...	36.0	6.0	2.0	[6.0,2.0]
Royal Challengers...	Chennai Super Kings	33.0	1.0	4.0	[1.0,4.0]
Kolkata Knight Ri...	Delhi Daredevils	31.0	2.0	6.0	[2.0,6.0]
Kings XI Punjab	Mumbai Indians	46.0	3.0	0.0	[3.0,0.0]
Chennai Super Kings	Delhi Capitals	58.0	4.0	10.0	[4.0,10.0]
Chennai Super Kings	Mumbai Indians	40.0	4.0	0.0	[4.0,0.0]
Rajasthan Royals	Chennai Super Kings	44.0	6.0	4.0	[6.0,4.0]
Mumbai Indians	Kolkata Knight Ri...	40.0	0.0	2.0	[0.0,2.0]
Kolkata Knight Ri...	Delhi Daredevils	54.0	2.0	6.0	[2.0,6.0]
Sunrisers Hyderabad	Kings XI Punjab	33.0	7.0	3.0	[7.0,3.0]
Kolkata Knight Ri...	Royal Challengers...	53.0	2.0	1.0	[2.0,1.0]
Chennai Super Kings	Mumbai Indians	90.0	4.0	0.0	[4.0,0.0]
Kolkata Knight Ri...	Delhi Daredevils	47.0	2.0	6.0	[2.0,6.0]

Figure 29:Assemble all the features with vector assembler

Splitting training data and test data

```
%pyspark
training_data, test_data = transformed_data.randomSplit([0.8, 0.2], seed = 1234)
print("Training Dataset Count: " + str(training_data.count()))
print("Test Dataset Count: " + str(test_data.count()))
```

```
Training Dataset Count: 1309
Test Dataset Count: 321
```

Figure 30:Splitting data for test and train

Training data with Gradient Booster Tree Regression model with iteration of 70

```
%pyspark
from pyspark.ml.regression import GBTRegressor
lr = GBTRegressor(featuresCol = 'features', labelCol='Power_play_score', maxIter=70)
lr_model = lr.fit(training_data)
```

Figure 31:Training data with Gradient Booster Tree Regression model with iteration of 70

Showing predictions and actual score of teams against their relevant opponent teams

```
%pyspark
lr_predictions = lr_model.transform(test_data)
lr_predictions.select("prediction","Power_play_score","features","batting_team","bowling_team").show()
from pyspark.ml.evaluation import RegressionEvaluator
lr_evaluator = RegressionEvaluator(predictionCol="prediction", \
                 labelCol="Power_play_score",metricName="r2")
print("R Squared (R2) on test data = %g" % lr_evaluator.evaluate(lr_predictions))
```

```
+------------------+---------------+----------+--------------------+--------------------+
|        prediction|Power_play_score|  features|        batting_team|        bowling_team|
+------------------+---------------+----------+--------------------+--------------------+
| 38.39051002955252|           58.0|[4.0,10.0]| Chennai Super Kings|      Delhi Capitals|
| 41.10751306287104|           40.0| [4.0,0.0]| Chennai Super Kings|      Mumbai Indians|
| 43.83649602426163|           60.0| [5.0,0.0]|    Delhi Daredevils|      Mumbai Indians|
| 41.10751306287104|           90.0| [4.0,0.0]| Chennai Super Kings|      Mumbai Indians|
| 48.23451372000446|           33.0| [7.0,3.0]| Sunrisers Hyderabad|     Kings XI Punjab|
|46.213990950876095|           57.0| [0.0,6.0]|      Mumbai Indians|    Delhi Daredevils|
| 47.23926464579712|           47.0| [1.0,5.0]|Royal Challengers...|    Rajasthan Royals|
| 45.21984853717641|           39.0| [5.0,4.0]|    Delhi Daredevils| Chennai Super Kings|
|48.484608014893446|           52.0| [2.0,4.0]|Kolkata Knight Ri...| Chennai Super Kings|
| 37.38280558921342|           31.0| [2.0,8.0]|Kolkata Knight Ri...|     Deccan Chargers|
| 47.30304365179937|           67.0| [2.0,7.0]|Kolkata Knight Ri...| Sunrisers Hyderabad|
|43.407848963599534|           68.0| [0.0,3.0]|      Mumbai Indians|     Kings XI Punjab|
| 47.05033476769016|           54.0| [5.0,1.0]|    Delhi Daredevils|Royal Challengers...|
| 50.87679337559307|           56.0| [7.0,1.0]| Sunrisers Hyderabad|Royal Challengers...|
```

Figure 32:Showing predictions and actual score of teams against their relevant opponent teams

4. Presentation of the analysis

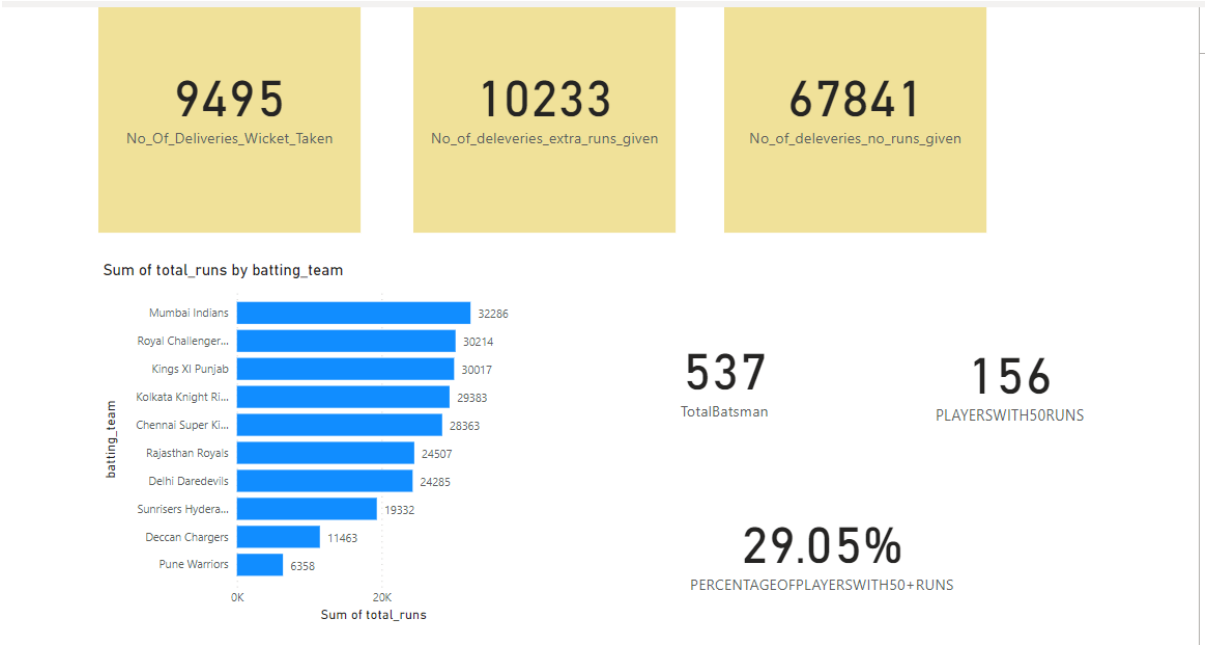


Figure 34:PowerBi Data Analysis for first three questions

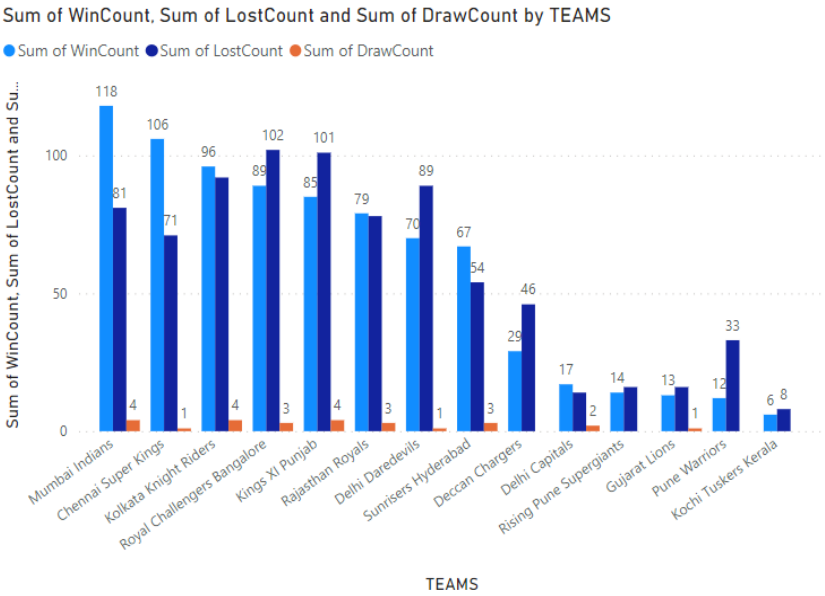


Figure 33:PowerBi fourth question

References

hokkaido university, (2019), containerbased-sizing-framework-for-apache-hadoopspark-clusters [ONLINE]. Available at: https://image.slidesharecdn.com/1110akiyoshisugikiv1-161031185742/95/a-containerbased-sizing-framework-for-apache-hadoopspark-clusters-10-638.jpg?cb=1477940273 [Accessed 10 January 2019].