<u>CMM705</u>

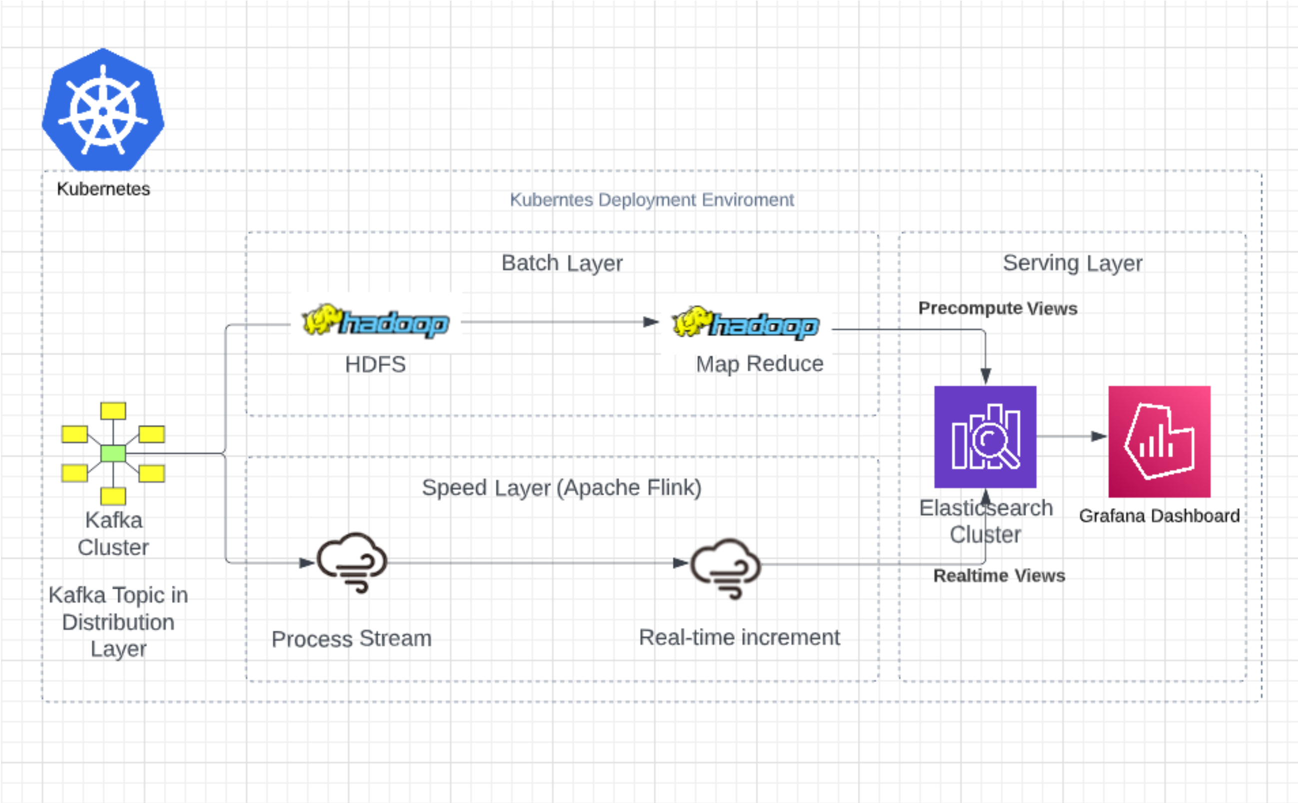
<u>COURSEWORK</u>

Name: G. A. J. M Senadeera

ID: 2121903 (RGU) / 20211475(IIT)

1. Solution Architecture

1. This system should have the ability to analyze real-time and historical data. So, I would recommend implementing the most prominent Lambda architecture because it supports both historical and real time data views to end users to query.



2. This proposed system is planned to deploy in a **Kubernetes** Deployment environment because **CNCF** supports each of these items to deploy in such environments (https://landscape.cncf.io/). And let's describe each section/component in this architecture.

Distribution Layer:

This layer mainly consisted of a **Kafka cluster**, this messaging broker is widely used in the industry because of multiple advantages of using it (performance, deployment support, features). In the broker, in this design, we are interested in a **single topic** which will be receiving real time cricket data. This data will be **multiplexed** into the speed and batch layers for processing.

Batch Layer:

This layer is considering processing legacy data that will be **gathered and accumulated** over time from the **Kafka topic** which we discussed earlier, and we call the data as **master dataset**, which is **immutable and incremental**. **Hadoop HDFS cluster** is used to save this dataset. And then **MapReduce** jobs will be triggered to processing on the dataset periodically. HDFS itself provides reliable and durable distributed datastore and it could manage to scale out over higher volumes of data received over the time.

Speed Layer:

This layer is mainly focused on processing real time data that has not been replicated in the **master dataset**. This incremental portion will be processed in real time using **Apache Flink** and saved in incremental views which is a new technology compared to **Apache Strom**, it provides high throughput and seamless integration with data sinks with simpler coding.

Serving Layer:

So far, we have discussed about two processing layers, but we didn't discuss where to point out the data after processing them. After each process/layer that we have discussed, the data will be saved in an **Elasticsearch**, NoSQL database. **Flink** natively supports **Elasticsearch** as a data **sink**, while we can configure **Hadoop MapReduce** Jobs to save processed data in the **Elasticsearch** by configuring **elasticsearch-hadoop.jar** and implementing the code which is provided by the **Elasticsearch team** to integrate with **Hadoop**.

Advantage of using this is, we will have single source of truth of the whole data that has been processed and further integration will also be simple with the other services because of this fact.

In the serving layer, we have also a **Grafana** deployment which will be looking at the **Elasticsearch** database and query data from it into its dashboard components. The reason for using this combination of **Elasticsearch and Grafana** is, they have seamless integration with each other. On the other hand, **Grafana** is widely used in the industry because it has mature features, and **Elasticsearch** is distributed analytics engine for free.

2. Data Analysis

2.1. Analyze the following using Hadoop MapReduce

For the analysis, I use **Kubernetes** cluster to deploy **hadoop-hive-pig container** because I don't have enough resources to run this container on my local machine. The following prerequisite steps would be the same for both **(2.1, 2.2)** questions.

i. First, switch the context to get into the correct cluster.

kubectl config use-context se-prod-k8s-user@se.prod.cluster.local

```
jmadushan@CLLK-JANITHAM:~$  kubectl config use-context se-prod-k8s-user@se.prod.cluster.local
Switched to context "se-prod-k8s-user@se.prod.cluster.local".
jmadushan@CLLK-JANITHAM:~$
```

ii. Create a pod using the docker image **suhothayan/hadoop-hive-pig:2.7.1**

kubectl run hdfs –image suhothayan/4adoop-hive-pig:2.7.1 –restart Never

```
jmadushan@CLLK-JANITHAM:~$ kubectl run hdfs --image suhothayan/hadoop-hive-pig:2.7.1 --restart Never
pod/hdfs created
jmadushan@CLLK-JANITHAM:~$
```

iii. Now copy the given **dataset** into the **pod/container** that has been created by the previous step.

kubectl cp ipl/ipl-data.csv hdfs:/

Let's see the copied content

kubectl exec -it hdfs – ls

```
jmadushan@CLLK-JANITHAM:~/rgu/CMM705$ kubectl exec -it hdfs -- ls
bin        dev   ipl-data.csv  media         opt   sbin     sys  var
boot       etc   lib           metastore_db  proc  selinux  tmp
derby.log  home  lib64         mnt           root  srv      usr
jmadushan@CLLK-JANITHAM:~/rgu/CMM705$
```

iv. Connect into the pod that we have created

kubectl exec -it hdfs – sh

```
jmadushan@CLLK-JANITHAM:~/rgu/CMM705$ kubectl exec -it hdfs -- sh
sh-4.1#
```

Create directory called **ipl** in the **hdfs**

hdfs dfs -mkdir /ipl

```
jmadushan@CLLK-JANITHAM:~/rgu/CMM705$ kubectl exec -it hdfs -- sh
sh-4.1# hdfs dfs -mkdir /ipl
22/12/05 11:57:05 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
sh-4.1#
```

And put the dataset **ipl-data.csv** into the folder that we have created in the **hdfs**

hdfs dfs -put ipl-data.csv /ipl/ipl-data.csv

```
sh-4.1# hdfs dfs -put ipl-data.csv /ipl/ipl-data.csv
22/12/05 11:59:17 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
sh-4.1#
```

MapReduce code has been included in the folder called **coursework-mapreduce-code.**

In order to run the map-reduce jobs, we need to build the related jar first using following command.
mvn clean package

```
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ coursework-mapreduce ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-jar-plugin:2.4:jar (default-jar) @ coursework-mapreduce ---
[INFO] Building jar: C:\workspace\iit\cmm705\cw\course-work-mapreduce\target\coursework-mapreduce-1.0-SNAPSHOT.jar
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  10.314 s
[INFO] Finished at: 2022-12-06T07:17:50+05:30
[INFO] ------------------------------------------------------------------------

Process finished with exit code 0
```

And copy the jar file **(coursework-mapreduce-1.0-SNAPSHOT.jar)** into the **hdfs** container that we have already created.

kubectl cp target/coursework-mapreduce-1.0-SNAPSHOT.jar hdfs:/

```
madushan@CLLK-JANITHAM:~/ws/iit/cmm705/cw/course-work-mapreduce$ kubectl cp target/
classes/                              generated-sources/                    maven-status/
coursework-mapreduce-1.0-SNAPSHOT.jar  maven-archiver/
madushan@CLLK-JANITHAM:~/ws/iit/cmm705/cw/course-work-mapreduce$ kubectl cp target/coursework-mapreduce-1.0-SNAPSHOT.jar hdfs:/
madushan@CLLK-JANITHAM:~/ws/iit/cmm705/cw/course-work-mapreduce$
```

The reduction implementation that I am going to use for both questions, is the same as the following mentioned. It will count the number of occurrences of each key and save write it back to the HDFS. ***The implementation is mostly the same that we have used in the lab sessions in the class.***

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

    public void reduce(
            final Text key,
            final Iterable<IntWritable> values,
            final Context context
    ) throws IOException, InterruptedException {
        // Count the occurrences with the same key
        int sum = 0;
        for (IntWritable val : values) {
            sum += val.get();
        }
        // Set the count to the result object
        result.set(sum);
        // Set output with the key
        context.write(key, result);
    }
}
```

1. Mapper implementation and the main method invoker for the question 1 would be,

```java
public class Q1 {

    public static class DeliveryMapper
            extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        @Override
        public void map(
                final Object key,
                final Text lineText,
                final Context context
        ) {
            try {
                final String line = lineText.toString();
                final String extraRuns = line.split(",")[8];
                final String isWicket = line.split(",")[11];
                final String totalRuns = line.split(",")[9];

                // If any extra runs were occurred count them
                if (Integer.parseInt(extraRuns) > 0) {
                    word.set("extraRuns");
                    context.write(word, one);
                }

                // If any wicket was taken count them
                if ("1".equals(isWicket)) {
                    word.set("wicketCount");
                    context.write(word, one);
                }

                // If any no runs were observed count them
                if ("0".equals(totalRuns)) {
                    word.set("noRuns");
                    context.write(word, one);
                }
            } catch (Exception e) {
                // There are some issues in the dataset causes number format exception
                // Ignoring them
            }
        }
    }

    public static void main(String[] args) throws Exception {
        // Invoker static method is mostly same as we had in the lab sessions
        // The full implementation can be seen in the zip file
    }
}
```

Now let's call the **yarn** with copied jar file to execute the map reduce job for the question 1 in the environment that we are already in.

yarn jar coursework-mapreduce-1.0-SNAPSHOT.jar mapreduce.Q1 /ipl /output

```
sh-4.1# yarn jar coursework-mapreduce-1.0-SNAPSHOT.jar mapreduce.Q1 /ipl /output
22/12/05 22:20:35 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
22/12/05 22:20:36 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
22/12/05 22:20:36 INFO input.FileInputFormat: Total input paths to process : 1
22/12/05 22:20:36 INFO mapreduce.JobSubmitter: number of splits:1
22/12/05 22:20:36 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1670256987498_0011
22/12/05 22:20:36 INFO impl.YarnClientImpl: Submitted application application_1670256987498_0011
22/12/05 22:20:36 INFO mapreduce.Job: The url to track the job: http://hdfs:8088/proxy/application_1670256987498_0011/
22/12/05 22:20:36 INFO mapreduce.Job: Running job: job_1670256987498_0011
22/12/05 22:20:41 INFO mapreduce.Job: Job job_1670256987498_0011 running in uber mode : false
22/12/05 22:20:41 INFO mapreduce.Job:  map 0% reduce 0%
22/12/05 22:20:47 INFO mapreduce.Job:  map 100% reduce 0%
22/12/05 22:20:53 INFO mapreduce.Job:  map 100% reduce 100%
22/12/05 22:20:53 INFO mapreduce.Job: Job job_1670256987498_0011 completed successfully
22/12/05 22:20:53 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=53
                FILE: Number of bytes written=231659
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=22721754
                HDFS: Number of bytes written=46
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=3339
                Total time spent by all reduces in occupied slots (ms)=3077
                Total time spent by all map tasks (ms)=3339
                Total time spent by all reduce tasks (ms)=3077
                Total vcore-seconds taken by all map tasks=3339
                Total vcore-seconds taken by all reduce tasks=3077
                Total megabyte-seconds taken by all map tasks=3419136
                Total megabyte-seconds taken by all reduce tasks=3150848
        Map-Reduce Framework
                Map input records=193469
                Map output records=87569
                Map output bytes=1041433
                Map output materialized bytes=53
                Input split bytes=98
                Combine input records=87569
                Combine output records=3
                Reduce input groups=3
                Reduce shuffle bytes=53
                Reduce input records=3
                Reduce output records=3
                Spilled Records=6
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=873
                CPU time spent (ms)=4100
                Physical memory (bytes) snapshot=432775168
                Virtual memory (bytes) snapshot=1728159744
                Total committed heap usage (bytes)=401604608
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
```

Now let's check the output.

hdfs dfs -cat /output/part-r-00000

```
sh-4.1# ^C
sh-4.1# hdfs dfs -ls /output
22/12/05 22:22:33 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 2 items
-rw-r--r--   1 root supergroup          0 2022-12-05 22:20 /output/_SUCCESS
-rw-r--r--   1 root supergroup         46 2022-12-05 22:20 /output/part-r-00000
sh-4.1# hdfs dfs -cat /output/part-r-00000
22/12/05 22:22:49 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
extraRuns       10233
noRuns  67841
wicketCount     9495
sh-4.1#
```

Now let's compose the output into a tabular format.

Variable	Description	Value
extraRuns	Extra Runs taken	10233
noRuns	No Runs were taken	67841
wicketCount	Wickets were taken	9495

2. Mapper Implementation and the main method for the question 2 would be,

```
public class Q2 {

    public static class WicketsMapper
            extends Mapper<Object, Text, Text, IntWritable> {

        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();

        @Override
        public void map(
                final Object key,
                final Text lineText,
                final Context context
        ) {
            final String line = lineText.toString();
            try {
                final String bowlingTeam = line.split(",")[17].toUpperCase().trim();
                final String isWicket = line.split(",")[11].trim();
                // If any wicket was taken recorded them with the team
                if ("1".equals(isWicket.trim())) {
                    word.set(bowlingTeam);
                    context.write(word, one);
                }
            } catch (Exception e) {
                e.printStackTrace();
                // Ignore any exception was occurred
            }
        }
    }

    public static void main(String[] args) throws Exception {
        // Invoker static method is mostly same as we had in the lab sessions
        // The full implementation can be seen in the zip file

    }
}
```

Now let's execute the Q2 class to obtain results in the jar file.

yarn jar coursework-mapreduce-1.0-SNAPSHOT.jar mapreduce.Q2 /ipl /output

```
Deleted /output
sh-4.1# yarn jar coursework-mapreduce-1.0-SNAPSHOT.jar mapreduce.Q2 /ipl /output
22/12/05 22:30:49 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
22/12/05 22:30:49 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
22/12/05 22:30:50 INFO input.FileInputFormat: Total input paths to process : 1
22/12/05 22:30:50 INFO mapreduce.JobSubmitter: number of splits:1
22/12/05 22:30:50 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1670256987498_0012
22/12/05 22:30:50 INFO impl.YarnClientImpl: Submitted application application_1670256987498_0012
22/12/05 22:30:50 INFO mapreduce.Job: The url to track the job: http://hdfs:8088/proxy/application_1670256987498_0012/
22/12/05 22:30:50 INFO mapreduce.Job: Running job: job_1670256987498_0012
22/12/05 22:30:55 INFO mapreduce.Job: Job job_1670256987498_0012 running in uber mode : false
22/12/05 22:30:55 INFO mapreduce.Job:  map 0% reduce 0%
22/12/05 22:31:01 INFO mapreduce.Job:  map 100% reduce 0%
22/12/05 22:31:06 INFO mapreduce.Job:  map 100% reduce 100%
22/12/05 22:31:06 INFO mapreduce.Job: Job job_1670256987498_0012 completed successfully
22/12/05 22:31:06 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=417
                FILE: Number of bytes written=232387
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=22721754
                HDFS: Number of bytes written=381
                HDFS: Number of read operations=6
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=3075
                Total time spent by all reduces in occupied slots (ms)=2214
                Total time spent by all map tasks (ms)=3075
                Total time spent by all reduce tasks (ms)=2214
                Total vcore-seconds taken by all map tasks=3075
                Total vcore-seconds taken by all reduce tasks=2214
                Total megabyte-seconds taken by all map tasks=3148800
                Total megabyte-seconds taken by all reduce tasks=2267136
        Map-Reduce Framework
                Map input records=193469
                Map output records=9495
                Map output bytes=237089
                Map output materialized bytes=417
                Input split bytes=98
                Combine input records=9495
                Combine output records=16
                Reduce input groups=16
                Reduce shuffle bytes=417
                Reduce input records=16
                Reduce output records=16
                Spilled Records=32
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=747
                CPU time spent (ms)=3550
                Physical memory (bytes) snapshot=425197568
                Virtual memory (bytes) snapshot=1727336448
                Total committed heap usage (bytes)=398983168
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
```

And let’s see the output in the HDFS.

hdfs dfs -cat /output/part-r-00000

```
sh-4.1#
sh-4.1# hdfs dfs -cat  /output/part-r-00000
22/12/06 10:07:58 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform..
"CHENNAI SUPER KINGS"   1089
"DECCAN CHARGERS"       454
"DELHI CAPITALS"        212
"DELHI DAREDEVILS"      900
"GUJARAT LIONS" 152
"KINGS XI PUNJAB"       1086
"KOCHI TUSKERS KERALA"  73
"KOLKATA KNIGHT RIDERS" 1068
"MUMBAI INDIANS"        1231
"PUNE WARRIORS" 236
"RAJASTHAN ROYALS"      920
"RISING PUNE SUPERGIANT"        109
"RISING PUNE SUPERGIANTS"       79
"ROYAL CHALLENGERS BANGALORE"   1109
"SUNRISERS HYDERABAD"   759
NA      18
sh-4.1#
```

And let's tabulate the results were taken,

Team	Wickets
CHENNAI SUPER KINGS	1089
DECCAN CHARGERS	454
DELHI CAPITALS	212
DELHI DAREDEVILS	900
GUJARAT LIONS	152
KINGS XI PUNJAB	1086
KOCHI TUSKERS KERALA	73
KOLKATA KNIGHT RIDERS	1068
MUMBAI INDIANS	1231
PUNE WARRIORS	236
RAJASTHAN ROYALS	920
RISING PUNE SUPERGIANT	109
RISING PUNE SUPERGIANTS	79
ROYAL CHALLENGERS BANGALORE	1109
SUNRISERS HYDERABAD	759
NA	18

2.2 Analyze the following using Hive or Pig

For this analysis I am using **Hive** because of its simplicity, and let's create an external table using the dataset that we have put in the **HDFS** using the following command to reuse the **HDFS** storage.

create external table ipl_external

(id int, inning int, over_playing int, ball int, batsman string, non_striker string, bowler string, batsman_runs int, extra_runs int, total_runs int, non_boundary int, is_wicket int, dismissal_kind string, player_dismissed string, fielder string, extras_type string, batting_team string, bowling_team string) row format delimited fields terminated by ',' LOCATION '/ipl/';

```
22/12/05 11:59:17 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
sh-4.1# hive

Logging initialized using configuration in jar:file:/usr/local/apache-hive-1.2.2-bin/lib/hive-common-1.2.2.jar!/hive-log4j.properties
hive> create external table ipl_external
    > (id int, inning int, over_playing int, ball int, batsman string, non_striker string, bowler string, batsman_runs int, extra_runs int, total_runs int, non_boundary int
, is_wicket int, dismissal_kind string, player_dismissed string, fielder string, extras_type string, batting_team string, bowling_team string)
    > row format delimited fields terminated by ',' LOCATION '/ipl/';
OK
Time taken: 0.652 seconds
hive>
```

1. Let's run the following query to find out the top 10 performing team in the hive terminal.

   select batting_team, sum(total_runs) as team_total_runs from ipl_external group by batting_team order by team_total_runs desc limit 10;

```
Time taken: 0.652 seconds
hive> select batting_team, sum(total_runs) as team_total_runs from ipl_external group by batting_team order by team_total_runs desc limit 10;
Query ID = root_20221205120538_e6b30725-0ba1-42db-aed4-1089ac76dfb8
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_1670256987498_0001, Tracking URL = http://hdfs:8088/proxy/application_1670256987498_0001/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1670256987498_0001
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-12-05 12:05:45,620 Stage-1 map = 0%,  reduce = 0%
2022-12-05 12:05:50,843 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.62 sec
2022-12-05 12:05:56,130 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 2.97 sec
MapReduce Total cumulative CPU time: 2 seconds 970 msec
Ended Job = job_1670256987498_0001
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_1670256987498_0002, Tracking URL = http://hdfs:8088/proxy/application_1670256987498_0002/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1670256987498_0002
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2022-12-05 12:06:06,673 Stage-2 map = 0%,  reduce = 0%
2022-12-05 12:06:10,935 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.08 sec
2022-12-05 12:06:16,142 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 2.8 sec
MapReduce Total cumulative CPU time: 2 seconds 800 msec
Ended Job = job_1670256987498_0002
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 2.97 sec   HDFS Read: 22730268 HDFS Write: 1130 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 2.8 sec   HDFS Read: 5817 HDFS Write: 264 SUCCESS
Total MapReduce CPU Time Spent: 5 seconds 770 msec
OK
"Mumbai Indians"        32259
"Royal Challengers Bangalore"   30183
"Kings XI Punjab"       29990
"Kolkata Knight Riders" 29357
"Chennai Super Kings"   28344
"Rajasthan Royals"      24480
"Delhi Daredevils"      24264
"Sunrisers Hyderabad"   19314
"Deccan Chargers"       11448
"Pune Warriors" 6348
Time taken: 38.333 seconds, Fetched: 10 row(s)
hive>
```

Result gives us the output of top ten teams as follows,

Ranking	Team Name	Score
1	Mumbai Indians	32259
2	Royal Challengers Bangalore	30183
3	Kings XI Punjab	29990
4	Kolkata Knight Riders	29357
5	Chennai Super Kings	28344
6	Rajasthan Royals	24480
7	Delhi Daredevils	24264
8	Sunrisers Hyderabad	19314
9	Deccan Chargers	11448
10	Pune Warriors	6348

2. let's run following query in the hive terminal to obtain average runs per over, per inning in the dataset.

   select id, inning, sum(total_runs)/20 as average_runs from ipl_external group by id, inning order by id, inning;

```
hive> SELECT id, inning, sum(total_runs)/20 as average_runs from ipl_external group by id, inning order by id, inning;
Query ID = root_20221209232758_efedca51-384c-4b4b-be26-7f04b644d1b5
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_1670256987498_0034, Tracking URL = http://hdfs:8088/proxy/application_1670256987498_0034/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1670256987498_0034
Hadoop job information for Stage-1: number of mappers: 1; number of reducers: 1
2022-12-09 23:28:03,235 Stage-1 map = 0%,  reduce = 0%
2022-12-09 23:28:08,391 Stage-1 map = 100%,  reduce = 0%, Cumulative CPU 1.84 sec
2022-12-09 23:28:14,672 Stage-1 map = 100%,  reduce = 100%, Cumulative CPU 4.26 sec
MapReduce Total cumulative CPU time: 4 seconds 260 msec
Ended Job = job_1670256987498_0034
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_1670256987498_0035, Tracking URL = http://hdfs:8088/proxy/application_1670256987498_0035/
Kill Command = /usr/local/hadoop/bin/hadoop job  -kill job_1670256987498_0035
Hadoop job information for Stage-2: number of mappers: 1; number of reducers: 1
2022-12-09 23:28:24,103 Stage-2 map = 0%,  reduce = 0%
2022-12-09 23:28:28,239 Stage-2 map = 100%,  reduce = 0%, Cumulative CPU 1.56 sec
2022-12-09 23:28:34,447 Stage-2 map = 100%,  reduce = 100%, Cumulative CPU 3.41 sec
MapReduce Total cumulative CPU time: 3 seconds 410 msec
Ended Job = job_1670256987498_0035
MapReduce Jobs Launched:
Stage-Stage-1: Map: 1  Reduce: 1   Cumulative CPU: 4.26 sec   HDFS Read: 22731603 HDFS Write: 49493 SUCCESS
Stage-Stage-2: Map: 1  Reduce: 1   Cumulative CPU: 3.41 sec   HDFS Read: 54149 HDFS Write: 22585 SUCCESS
Total MapReduce CPU Time Spent: 7 seconds 670 msec
OK
NULL    NULL    NULL
335982  1       11.1
335982  2       4.1
335983  1       12.0
335983  2       10.35
335984  1       6.45
335984  2       6.6
335985  1       8.25
335985  2       8.3
335986  1       5.5
335986  2       5.6
335987  1       8.3
335987  2       8.4
335988  1       7.1
335988  2       7.15
335989  1       10.4
335989  2       10.1
335990  1       10.7
335990  2       10.85
335991  1       9.1
335991  2       5.8
```

The result is so lengthy. So, here I have included the topmost part of the output, and the columns are showing as **null** for some reason they should be **id, inning, average score** accordingly but the output result seems accurate.

2.3 Analyze the following using Spark.

First of all we need to run the zeppelin detached container with a mounted volume which contains **ipl-data.csv** dataset, and then import the notebook file **CMM-705_CW_final.zpln** in the UI of zeppelin web interface which is exposed in local port **8080**

docker run -it --name zeppelin -p 8080:8080 -v /home/jmadushan/rgu/CMM705/cw/ipl:/data apache/zeppelin:0.9.0

```
madushan@CLLK-JANITHAM:~$ docker run -it --name zeppelin -d -p 8080:8080 -v /home/jmadushan/rgu/CMM705/cw/ipl:/data apache/zeppelin:0.9.0
d14996dad36422c2be8fd3bac5df22cc5768d897c186725e54c96ccddc5d6f1
madushan@CLLK-JANITHAM:~$
```

First, we need to import the dataset from the mounted location and load it into the data frame.

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

df.show(10)

df.createOrReplaceTempView("batting")

```
%pyspark

# Load CSV file into the dataframe
df = spark.read.option("header",True).csv("/data/ipl-data.csv")

df.show(10)

# Create a view called batting
df.createOrReplaceTempView("batting")
```

```
+------+------+----+----+----------+-----------+---------+-------------+-----------+----------+------------+---------+--------------+----------------+------+-----------+--------------------+--------------------+
|    id|inning|over|ball|   batsman|non_striker|   bowler|batsman_runs|extra_runs|total_runs|non_boundary|is_wicket|dismissal_kind|player_dismissed|fielder|extras_type|        batting_team|        bowling_team|
+------+------+----+----+----------+-----------+---------+-------------+-----------+----------+------------+---------+--------------+----------------+------+-----------+--------------------+--------------------+
|335982|     1|   6|   5| RT Ponting|BB McCullum|AA Noffke|            1|          0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   6|   6|BB McCullum| RT Ponting|AA Noffke|            1|          0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   7|   1|BB McCullum| RT Ponting|   Z Khan|            0|          0|         0|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   7|   2|BB McCullum| RT Ponting|   Z Khan|            1|          0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   7|   3| RT Ponting|BB McCullum|   Z Khan|            1|          0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   7|   4|BB McCullum| RT Ponting|   Z Khan|            1|          0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   7|   5| RT Ponting|BB McCullum|   Z Khan|            1|          0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   7|   6|BB McCullum| RT Ponting|   Z Khan|            1|          0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   8|   1|BB McCullum| RT Ponting|JH Kallis|            0|          0|         0|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   8|   2|BB McCullum| RT Ponting|JH Kallis|            0|          0|         0|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
+------+------+----+----+----------+-----------+---------+-------------+-----------+----------+------------+---------+--------------+----------------+------+-----------+--------------------+--------------------+
only showing top 10 rows
```

1. In the above step we have created a view called `**batting**` on the main dataset, and now let's filter out/aggregate the data to give an output of **each player's total score per inning**.

   # Create a summary query to find total runs of each team member for an inning

   inning_df = spark.sql("SELECT id, inning, batsman, sum(total_runs) as total_runs_per_inning FROM batting group by id, inning, batsman")

   inning_df.show(10)

```
%pyspark

# Create a summary query to find total runs of each team member for an inning
inning_df = spark.sql("SELECT id, inning, batsman, sum(total_runs) as total_runs_per_inning FROM batting group by id, inning, batsman")
inning_df.show(10)
```

```
+------+------+---------+---------------------+
|    id|inning|  batsman|total_runs_per_inning|
+------+------+---------+---------------------+
|336029|     1|  P Kumar|                 21.0|
|336037|     1|SB Styris|                 24.0|
|392190|     2| CH Gayle|                 48.0|
|392196|     1| JDP Oram|                 42.0|
|392208|     2|RV Uthappa|                68.0|
|392218|     2| R Dravid|                 12.0|
|392220|     1| DR Smith|                 48.0|
|392230|     2| MM Patel|                 23.0|
|419124|     2| MS Bisla|                 37.0|
|419128|     2| R Dravid|                  1.0|
+------+------+---------+---------------------+
only showing top 10 rows
```

Took 1 sec. Last updated by anonymous at December 10 2022, 1:43:59 PM.

Now let's summarize **the first question** by performing operations on RDD of above created summarized **inning_df** data frame.

 # Count total number of players

total_players = df.rdd.map(lambda row: row.batsman).distinct().count()

 # Count total number of players scored more than 50 per inning

players_have_scored_more_than_50 = inning_df.rdd.filter(lambda p: p.total_runs_per_inning>50).map(lambda inning: inning.batsman).distinct().count()

print("Total number of players: %d\n" % total_players)

print("Number of players who have scored more than 50 per inning is : %d\n" % players_have_scored_more_than_50)

print("Percentage of players who have scored more than 50 per inning is : %.2f " % (players_have_scored_more_than_50/total_players*100))

```
%pyspark

# Count total number of players
total_players = df.rdd.map(lambda row: row.batsman).distinct().count()

# Count total number of players scored more than 50 per inning
players_have_scored_more_than_50 = inning_df.rdd.filter(lambda p: p.total_runs_per_inning>50).map(lambda inning: inning.batsman).distinct().count()

print("Total number of players: %d\n" % total_players)
print("Number of players who have scored more than 50 per inning is : %d\n" % players_have_scored_more_than_50)
print("Percentage of players who have scored more than 50 per inning is :  %.2f " % (players_have_scored_more_than_50/total_players*100))

Total number of players: 537

Number of players who have scored more than 50 per inning is : 157

Percentage of players who have scored more than 50 per inning is :  29.24
```

Took 4 sec. Last updated by anonymous at December 10 2022, 1:44:11 PM.

Results obtained by the above analysis,

Description	Result
Total Number of distinct Players	537
Total Number of Players have scored more than 50 in a single inning	157
Percentage of players who have scored more than 50	29.24

2. Let's filter out the necessary fields from the main dataset in the data frame **df** and filter out the only the fields that are required to answer this question **(id, total_runs, batting_team, bowling_team)**.

selected_df = df.select(df['id'], df['total_runs'], df['batting_team'], df['bowling_team'])

selected_df.show(10)

```
%pyspark

# First select only the fields that's need for the analysis from the original dataframe
selected_df = df.select(df['id'], df['total_runs'], df['batting_team'], df['bowling_team'])
selected_df.show(10)

+------+----------+-------------------+-------------------+
|    id|total_runs|       batting_team|       bowling_team|
+------+----------+-------------------+-------------------+
|335982|         1|Kolkata Knight Ri...|Royal Challengers...|
|335982|         1|Kolkata Knight Ri...|Royal Challengers...|
|335982|         0|Kolkata Knight Ri...|Royal Challengers...|
|335982|         1|Kolkata Knight Ri...|Royal Challengers...|
|335982|         1|Kolkata Knight Ri...|Royal Challengers...|
|335982|         1|Kolkata Knight Ri...|Royal Challengers...|
|335982|         1|Kolkata Knight Ri...|Royal Challengers...|
|335982|         1|Kolkata Knight Ri...|Royal Challengers...|
|335982|         0|Kolkata Knight Ri...|Royal Challengers...|
|335982|         0|Kolkata Knight Ri...|Royal Challengers...|
+------+----------+-------------------+-------------------+
only showing top 10 rows
```

Now let's create a view called **'match'** on the data frame **selected_df,** and let's perform an SQL operation to find out total runs of each team as batting team in the each match.

 # create batting view called `match` for the selected columns

selected_df.createOrReplaceTempView("match")

 # create summary dataframe from each match and scores for batting and bowling teams.

match_summary_df = spark.sql("SELECT id, sum(total_runs) as total_runs, batting_team, bowling_team FROM match group by id, batting_team, bowling_team order by id")

match_summary_df.show(10)

```
%pyspark

# create batting view called `match` for the selected columns
selected_df.createOrReplaceTempView("match")

# create summary dataframe from each match and scores for batting and bowling teams.
match_summary_df = spark.sql("SELECT id, sum(total_runs) as total_runs, batting_team, bowling_team FROM match group by id, batting_team, bowling_team order by id")
match_summary_df.show(10)

+-------+----------+-------------------+-------------------+
|     id|total_runs|       batting_team|       bowling_team|
+-------+----------+-------------------+-------------------+
|1082591|     172.0|Royal Challengers...| Sunrisers Hyderabad|
|1082591|     207.0| Sunrisers Hyderabad|Royal Challengers...|
|1082592|     184.0|      Mumbai Indians|Rising Pune Super...|
|1082592|     187.0|Rising Pune Super...|      Mumbai Indians|
|1082593|     184.0|Kolkata Knight Ri...|       Gujarat Lions|
|1082593|     183.0|       Gujarat Lions|Kolkata Knight Ri...|
|1082594|     164.0|     Kings XI Punjab|Rising Pune Super...|
|1082594|     163.0|Rising Pune Super...|     Kings XI Punjab|
|1082595|     157.0|Royal Challengers...|    Delhi Daredevils|
|1082595|     142.0|    Delhi Daredevils|Royal Challengers...|
+-------+----------+-------------------+-------------------+
only showing top 10 rows
```

In the above output we can see match id is duplicated by 2 rows, means for each match it contains rows for each team as **batting team** and corresponding **bowling team**.

Row => [id], [total_runs_team_A, total_runs_team_B], [Team_A, Team_B]

And now let's perform aggregate function to aggregate columns to describe each match by a single row. So, aggregate functions have been called **total_runs, batting_team**, and **bowling_team** column as it is explained above.

from pyspark.sql.functions import collect_list, col

 # Now lets aggregate scores and corresponding teams of each match, and this would be the base dataframe to answer the questions.

match_scores_df = match_summary_df.groupBy("id").agg(collect_list(col("total_runs")).alias("scores"), collect_list(col("batting_team")).alias("teams"))

match_scores_df.show(10)

```
%pyspark

from pyspark.sql.functions import collect_list, col

# Now lets aggregate scores and corresponding teams of each match, and this would be the base dataframe to answer the questions.
match_scores_df = match_summary_df.groupBy("id").agg(collect_list(col("total_runs")).alias("scores"), collect_list(col("batting_team")).alias("teams"))
match_scores_df.show(10)

+-------+--------------+--------------------+
|     id|        scores|               teams|
+-------+--------------+--------------------+
|1082591|[207.0, 172.0]|[Sunrisers Hydera...|
|1082592|[187.0, 184.0]|[Rising Pune Supe...|
|1082593|[184.0, 183.0]|[Kolkata Knight R...|
|1082594|[164.0, 163.0]|[Kings XI Punjab,...|
|1082595|[142.0, 157.0]|[Delhi Daredevils...|
|1082596|[140.0, 135.0]|[Sunrisers Hydera...|
|1082597|[180.0, 178.0]|[Mumbai Indians, ...|
|1082598|[150.0, 148.0]|[Kings XI Punjab,...|
|1082599|[205.0, 108.0]|[Delhi Daredevils...|
|1082600|[158.0, 159.0]|[Sunrisers Hydera...|
+-------+--------------+--------------------+
only showing top 10 rows
```

Analize data for won matches.

And now let's add an additional column mentioning **which team has won** each game.

from pyspark.sql import functions as f

winners_df=match_scores_df[match_scores_df['scores'][0]!=match_scores_df['scores'][1]].withColumn('winners', f.when(f.col('scores')[0] > f.col('scores')[1], f.col('teams')[0]).otherwise(f.col('teams')[1]))

winners_df.show(10)

```
%pyspark

from pyspark.sql import functions as f

# Now lets compare the batting scores and add winners column
winners_df=match_scores_df[match_scores_df['scores'][0]!=match_scores_df['scores'][1]].withColumn('winners', f.when(f.col('scores')[0] > f.col('scores')[1], f.col('teams')[0]).otherwise(f.col('teams')[1]))
winners_df.show(10)

+-------+--------------+--------------------+--------------------+
|     id|        scores|               teams|             winners|
+-------+--------------+--------------------+--------------------+
|1082591|[207.0, 172.0]|[Sunrisers Hydera...| Sunrisers Hyderabad|
|1082592|[187.0, 184.0]|[Rising Pune Supe...|Rising Pune Super...|
|1082593|[184.0, 183.0]|[Kolkata Knight R...|Kolkata Knight Ri...|
|1082594|[164.0, 163.0]|[Kings XI Punjab,...|     Kings XI Punjab|
|1082595|[142.0, 157.0]|[Delhi Daredevils...|Royal Challengers...|
|1082596|[140.0, 135.0]|[Sunrisers Hydera...| Sunrisers Hyderabad|
|1082597|[180.0, 178.0]|[Mumbai Indians, ...|      Mumbai Indians|
|1082598|[150.0, 148.0]|[Kings XI Punjab,...|     Kings XI Punjab|
|1082599|[205.0, 108.0]|[Delhi Daredevils...|    Delhi Daredevils|
|1082600|[158.0, 159.0]|[Sunrisers Hydera...|      Mumbai Indians|
+-------+--------------+--------------------+--------------------+
only showing top 10 rows
```

Finally, let's create a view called `**winners**` on the **winners_df** that we created in the previous step, and summarize each team's total number of winning matches.

winners_df.createOrReplaceTempView("winners")

winners_summary_df = spark.sql("SELECT winners as team, count(*) as wins FROM winners group by winners")

winners_summary_df.show(10)

```
%pyspark
# Create view as winners
winners_df.createOrReplaceTempView("winners")

# Summarize each teams winning count
winners_summary_df = spark.sql("SELECT winners as team, count(*) as wins FROM winners group by winners")
winners_summary_df.show(10)

+--------------------+----+
|                team|wins|
+--------------------+----+
| Sunrisers Hyderabad|  67|
| Chennai Super Kings| 106|
|Rising Pune Super...|  10|
|     Deccan Chargers|  29|
|Kochi Tuskers Kerala|   6|
|    Rajasthan Royals|  79|
|       Gujarat Lions|  13|
|Royal Challengers...|  89|
|Kolkata Knight Ri...|  96|
|Rising Pune Super...|   4|
+--------------------+----+
only showing top 10 rows
```

Analyze data for matches lost.

Next, let's create another data frame called **loosers_df** containing an additional column mentioning the losing team.

loosers_df=match_scores_df[match_scores_df['scores'][0]!=match_scores_df['scores'][1]].withColumn('loosers', f.when(f.col('scores')[0] > f.col('scores')[1], f.col('teams')[1]).otherwise(f.col('teams')[0]))

loosers_df.show(10)

```
%pyspark

# Summarize loosing team for each match
loosers_df=match_scores_df[match_scores_df['scores'][0]!=match_scores_df['scores'][1]].withColumn('loosers', f.when(f.col('scores')[0] > f.col('scores')[1], f.col('teams')[1]).otherwise(f.col('teams')[0]))
loosers_df.show(10)

+-------+------------+--------------------+--------------------+
|     id|      scores|               teams|             loosers|
+-------+------------+--------------------+--------------------+
|1082591|[207.0, 172.0]|[Sunrisers Hydera...|Royal Challengers...|
|1082592|[187.0, 184.0]|[Rising Pune Supe...|      Mumbai Indians|
|1082593|[184.0, 183.0]|[Kolkata Knight R...|       Gujarat Lions|
|1082594|[164.0, 163.0]|[Kings XI Punjab,...|Rising Pune Super...|
|1082595|[142.0, 157.0]|[Delhi Daredevils...|    Delhi Daredevils|
|1082596|[140.0, 135.0]|[Sunrisers Hydera...|       Gujarat Lions|
|1082597|[180.0, 178.0]|[Mumbai Indians, ...|Kolkata Knight Ri...|
|1082598|[150.0, 148.0]|[Kings XI Punjab,...|Royal Challengers...|
|1082599|[205.0, 108.0]|[Delhi Daredevils...|Rising Pune Super...|
|1082600|[158.0, 159.0]|[Sunrisers Hydera...| Sunrisers Hyderabad|
+-------+------------+--------------------+--------------------+
only showing top 10 rows
```

And create a view called losers mentioning each team losing matches.

loosers_df.createOrReplaceTempView("loosers")

loosers_summary_df = spark.sql("SELECT loosers as team, count(*) as losses FROM loosers group by loosers")

loosers_summary_df.show(10)

```
%pyspark
# create a view called loosers
loosers_df.createOrReplaceTempView("loosers")

# summarize loosing count for each team
loosers_summary_df = spark.sql("SELECT loosers as team, count(*) as losses FROM loosers group by loosers")
loosers_summary_df.show(10)

+--------------------+------+
|                team|losses|
+--------------------+------+
| Sunrisers Hyderabad|    54|
| Chennai Super Kings|    71|
|Rising Pune Super...|     6|
|     Deccan Chargers|    46|
|Kochi Tuskers Kerala|     8|
|    Rajasthan Royals|    78|
|       Gujarat Lions|    16|
|Royal Challengers...|   102|
|Kolkata Knight Ri...|    92|
|Rising Pune Super...|    10|
+--------------------+------+
only showing top 10 rows
```

Analyze data for matches drawn.

And now let's find the occurrences that each team has drawn.

eq_df=match_scores_df[match_scores_df['scores'][0]==match_scores_df['scores'][1]]

```
%pyspark

# filter the matches has been drawn
eq_df=match_scores_df[match_scores_df['scores'][0]==match_scores_df['scores'][1]]

eq_selected_df = eq_df.select(['teams'])

# Splitup the teams of lists
eq_selected_rdd=eq_selected_df.rdd.flatMap(lambda x:x).flatMap(lambda x:x)

# create a datafram as draw with splitted team names
draw_df = eq_selected_rdd.map(lambda x: (x, )).toDF(["draw"])
draw_df.show(10)


+--------------------+
|                draw|
+--------------------+
|       Gujarat Lions|
|      Mumbai Indians|
|      Delhi Capitals|
|Kolkata Knight Ri...|
| Sunrisers Hyderabad|
|      Mumbai Indians|
|     Kings XI Punjab|
|      Delhi Capitals|
|Kolkata Knight Ri...|
| Sunrisers Hyderabad|
+--------------------+
only showing top 10 rows
```

eq_selected_df = eq_df.select(['teams'])

eq_selected_rdd=eq_selected_df.rdd.flatMap(lambda x:x).flatMap(lambda x:x)

draw_df = eq_selected_rdd.map(lambda x: (x,)).toDF(["draw"])

draw_df.show(10)

And now let's create a view on **drawn_df** called **`draw`** to summarize the total number of matches that have been drawn by each team.

draw_df.createOrReplaceTempView("draws")

draw_summary_df = spark.sql("SELECT draw as team, count(*) as draws FROM draws group by draw")draw_summary_df.show(10)

```
%pyspark
# Create a dataframe called draws
draw_df.createOrReplaceTempView("draws")

# Summarize each teams drawn count
draw_summary_df = spark.sql("SELECT draw as team, count(*) as draws FROM draws group by draw")
draw_summary_df.show(10)

+--------------------+-----+
|                team|draws|
+--------------------+-----+
| Sunrisers Hyderabad|    3|
| Chennai Super Kings|    1|
|    Rajasthan Royals|    3|
|       Gujarat Lions|    1|
|Royal Challengers...|    3|
|Kolkata Knight Ri...|    4|
|     Kings XI Punjab|    4|
|    Delhi Daredevils|    1|
|      Delhi Capitals|    2|
|      Mumbai Indians|    4|
+--------------------+-----+
```

And let’s aggregate all outputs into a single data frame to summarize each team’s performance.

summary_df=winners_summary_df.join(loosers_summary_df, ["team"]).join(draw_summary_df, ["team"])

summary_df.show()

```
%pyspark

# Aggregate all the datafrmes together to summarize teach teams statistics
summary_df=winners_summary_df.join(loosers_summary_df, ["team"]).join(draw_summary_df, ["team"])
summary_df.show()

+--------------------+----+------+-----+
|                team|wins|losses|draws|
+--------------------+----+------+-----+
| Sunrisers Hyderabad|  67|    54|    3|
| Chennai Super Kings| 106|    71|    1|
|    Rajasthan Royals|  79|    78|    3|
|       Gujarat Lions|  13|    16|    1|
|Royal Challengers...|  89|   102|    3|
|Kolkata Knight Ri...|  96|    92|    4|
|     Kings XI Punjab|  85|   101|    4|
|    Delhi Daredevils|  70|    89|    1|
|      Delhi Capitals|  17|    14|    2|
|      Mumbai Indians| 118|    81|    4|
+--------------------+----+------+-----+
```

Finally, let’s export **summary_df** into a csv file to import in the dashboard.

summary_df.coalesce(1).write.options(header='True', delimiter=',') .format("csv").mode('overwrite').csv("/data/summary")

3. Performing Machine Learning model using Spark MLlib

We have already loaded the dataset into a data frame **df** and created a view called **batting** and let's fetch the first ten columns of that view.

spark.sql("SELECT * FROM batting").show(10)

```
%pyspark
spark.sql("SELECT * FROM batting").show(10)

+------+------+----+----+-----------+-----------+---------+------------+----------+----------+------------+---------+--------------+----------------+------+-----------+-------------------+-------------------+
|    id|inning|over|ball|    batsman|non_striker|   bowler|batsman_runs|extra_runs|total_runs|non_boundary|is_wicket|dismissal_kind|player_dismissed|fielder|extras_type|       batting_team|       bowling_team|
+------+------+----+----+-----------+-----------+---------+------------+----------+----------+------------+---------+--------------+----------------+------+-----------+-------------------+-------------------+
|335982|     1|   6|   5| RT Ponting|BB McCullum|AA Noffke|           1|         0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   6|   6|BB McCullum| RT Ponting|AA Noffke|           1|         0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   7|   1|BB McCullum| RT Ponting|   Z Khan|           0|         0|         0|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   7|   2|BB McCullum| RT Ponting|   Z Khan|           1|         0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   7|   3| RT Ponting|BB McCullum|   Z Khan|           1|         0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   7|   4|BB McCullum| RT Ponting|   Z Khan|           1|         0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   7|   5| RT Ponting|BB McCullum|   Z Khan|           1|         0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   7|   6|BB McCullum| RT Ponting|   Z Khan|           1|         0|         1|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   8|   1|BB McCullum| RT Ponting|JH Kallis|           0|         0|         0|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
|335982|     1|   8|   2|BB McCullum| RT Ponting|JH Kallis|           0|         0|         0|           0|        0|            NA|              NA|    NA|         NA|Kolkata Knight Ri...|Royal Challengers...|
+------+------+----+----+-----------+-----------+---------+------------+----------+----------+------------+---------+--------------+----------------+------+-----------+-------------------+-------------------+
only showing top 10 rows
```

let's analyze **overs** column,

spark.sql("SELECT distinct over FROM batting order by over asc").show()

```
%pyspark
spark.sql("SELECT distinct over FROM batting order by over asc").show()

+----+
|over|
+----+
|   0|
|   1|
|  10|
|  11|
|  12|
|  13|
|  14|
|  15|
|  16|
|  17|
|  18|
|  19|
|   2|
|   3|
```

Here we can see the overs are starting from zero. So, I assume **0** as the **first over**, and sum up each team's total score over the matches as batting teams in the first 6 overs. The assignment says Build a model that predicts the average runs expected to be scored in the first 6 overs if **TeamA** team plays against **TeamB**. So, this means we need to select **batting_team, bowling_team** as features, and sum of runs as the target variable.

batting_summary_df = spark.sql("SELECT id, batting_team, bowling_team, sum(total_runs) as sum FROM batting where over < 6 group by id, batting_team, bowling_team order by id")

batting_summary_df.show(10)

```
%pyspark

batting_summary_df = spark.sql("SELECT id, batting_team, bowling_team, sum(total_runs) as sum FROM batting where over < 6 group by id, batting_team, bowling_team order by id")
batting_summary_df.show(10)

+-------+-------------------+-------------------+----+
|     id|       batting_team|       bowling_team| sum|
+-------+-------------------+-------------------+----+
|1082591| Sunrisers Hyderabad|Royal Challengers...|59.0|
|1082591|Royal Challengers...| Sunrisers Hyderabad|54.0|
|1082592|Rising Pune Super...|      Mumbai Indians|59.0|
|1082592|      Mumbai Indians|Rising Pune Super...|61.0|
|1082593|       Gujarat Lions|Kolkata Knight Ri...|52.0|
|1082593|Kolkata Knight Ri...|       Gujarat Lions|73.0|
|1082594|     Kings XI Punjab|Rising Pune Super...|56.0|
|1082594|Rising Pune Super...|     Kings XI Punjab|35.0|
|1082595|    Delhi Daredevils|Royal Challengers...|43.0|
|1082595|Royal Challengers...|    Delhi Daredevils|41.0|
+-------+-------------------+-------------------+----+
only showing top 10 rows

Took 1 sec. Last updated by anonymous at December 10 2022, 4:12:02 PM.
```

Ok, now we have obtained each team score for the first 6 overs along with the bowling team, but in machine learning, it is necessary to code categorical variables with numeric interpretation, called categorical coding. So, let's perform categorical coding over **batting_team**, and bowling team variables as **batting_coded & bowling_coded** variables.

from pyspark.ml.feature import StringIndexer, OneHotEncoder

 # create indexers

batting_indexer=StringIndexer(inputCol='batting_team', outputCol='batting_coded')

bowling_indexer=StringIndexer(inputCol='bowling_team', outputCol='bowling_coded')

#perform categorical coding

indexer_batting = batting_indexer.fit(batting_summary_df)

indexed_df = indexer_batting.transform(batting_summary_df)

indexer_bowling = bowling_indexer.fit(indexed_df)

indexed_df = indexer_bowling.transform(indexed_df)

indexed_df.show()

```
%pyspark
#https://masum-math8065.medium.com/how-to-convert-categorical-data-into-numeric-in-pyspark-2202407f5fac
from pyspark.ml.feature import StringIndexer, OneHotEncoder

# create indexers
batting_indexer=StringIndexer(inputCol='batting_team', outputCol='batting_coded')
bowling_indexer=StringIndexer(inputCol='bowling_team', outputCol='bowling_coded')

#perform categorical coding
indexer_batting = batting_indexer.fit(batting_summary_df)
indexed_df = indexer_batting.transform(batting_summary_df)
indexer_bowling = bowling_indexer.fit(indexed_df)
indexed_df = indexer_bowling.transform(indexed_df)

indexed_df.show()

+-------+--------------------+--------------------+----+-------------+-------------+
|     id|        batting_team|        bowling_team| sum|batting_coded|bowling_coded|
+-------+--------------------+--------------------+----+-------------+-------------+
| 392211|Kolkata Knight Ri...|    Delhi Daredevils|54.0|          2.0|          6.0|
| 419148|Royal Challengers...|Kolkata Knight Ri...|48.0|          1.0|          2.0|
| 501218|       Pune Warriors|      Mumbai Indians|33.0|          9.0|          0.0|
|1082643|       Gujarat Lions| Sunrisers Hyderabad|61.0|         11.0|          7.0|
|1178429| Sunrisers Hyderabad|Royal Challengers...|52.0|          7.0|          1.0|
|1216531|     Kings XI Punjab|Royal Challengers...|56.0|          3.0|          1.0|
| 392212|     Deccan Chargers|      Mumbai Indians|37.0|          8.0|          0.0|
| 392238|Royal Challengers...| Chennai Super Kings|59.0|          1.0|          4.0|
| 501202|     Kings XI Punjab|       Pune Warriors|36.0|          3.0|          9.0|
| 501219|Kochi Tuskers Kerala|Kolkata Knight Ri...|45.0|         13.0|          2.0|
| 501234|     Kings XI Punjab|Kolkata Knight Ri...|40.0|          3.0|          2.0|
| 598002| Chennai Super Kings|      Mumbai Indians|36.0|          4.0|          0.0|
| 598036|Royal Challengers...|    Rajasthan Royals|55.0|          1.0|          5.0|
| 598045|Royal Challengers...|     Kings XI Punjab|31.0|          1.0|          3.0|
```

Now let's filter out only the necessary variables in the **indexed_df.**

coded_df=indexed_df.select("batting_coded", "bowling_coded", "sum")

coded_df.show()

```
%pyspark
coded_df=indexed_df.select("batting_coded", "bowling_coded", "sum")
coded_df.show()

+-------------+-------------+----+
|batting_coded|bowling_coded| sum|
+-------------+-------------+----+
|          2.0|          6.0|54.0|
|          1.0|          2.0|48.0|
|          9.0|          0.0|33.0|
|         11.0|          7.0|61.0|
|          7.0|          1.0|52.0|
|          3.0|          1.0|56.0|
|          8.0|          0.0|37.0|
|          1.0|          4.0|59.0|
|          3.0|          9.0|36.0|
|         13.0|          2.0|45.0|
|          3.0|          2.0|40.0|
|          4.0|          0.0|36.0|
|          1.0|          5.0|55.0|
|          1.0|          3.0|31.0|
```

Now let's prepare our final dataset to perform machine learning operations. I am using **VectorAssembler** in **PySpark** to create feature vectors.

from pyspark.ml.regression import LinearRegression

from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols=[

'batting_coded',

'bowling_coded'],

outputCol='features')

data_set = assembler.transform(coded_df)

data_set.show(2)

```
%pyspark
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler

assembler = VectorAssembler(inputCols=[
 'batting_coded',
 'bowling_coded'],
 outputCol='features')

data_set = assembler.transform(coded_df)
data_set.show(2)

+-------------+-------------+----+---------+
|batting_coded|bowling_coded| sum| features|
+-------------+-------------+----+---------+
|          2.0|          6.0|54.0|[2.0,6.0]|
|          1.0|          2.0|48.0|[1.0,2.0]|
+-------------+-------------+----+---------+
only showing top 2 rows
```

Let's filter out only the necessary fields from the above dataset.

```
%pyspark

final_df=data_set.select(['features','sum'])
final_df.show()

+----------+----+
|  features| sum|
+----------+----+
| [2.0,6.0]|54.0|
| [1.0,2.0]|48.0|
| [9.0,0.0]|33.0|
|[11.0,7.0]|61.0|
| [7.0,1.0]|52.0|
| [3.0,1.0]|56.0|
| [8.0,0.0]|37.0|
| [1.0,4.0]|59.0|
| [3.0,9.0]|36.0|
|[13.0,2.0]|45.0|
| [3.0,2.0]|40.0|
| [4.0,0.0]|36.0|
| [1.0,5.0]|55.0|
| [1.0,3.0]|31.0|
```

And now we need to split our final dataset into `80%` as tesing dataset and `20%` as the testing dataset.

train_data,test_data = final_df.randomSplit([0.8,0.2])

```
%pyspark
train_data,test_data = final_df.randomSplit([0.8,0.2])
```

Took 0 sec. Last updated by anonymous at December 05 2022, 9:15:41 PM.

I am going to use **Linear Regression model** in this case and fit in the `train_data` dataset to train the model.

from pyspark.ml.regression import LinearRegression

linearRegression = LinearRegression(labelCol='sum')

linearModel = linearRegression.fit(train_data)

```
%pyspark
from pyspark.ml.regression import LinearRegression

linearRegression = LinearRegression(labelCol='sum')
linearModel = linearRegression.fit(train_data)
```

Took 3 sec. Last updated by anonymous at December 05 2022, 9:19:16 PM.

Let us obtain parameters to evaluate the model accuracy, such as `RMSE` and `R2` with the `test_data` dataset.

test_stats = linearModel.evaluate(test_data)

print("RMSE:", test_stats.rootMeanSquaredError)

print("R2: ", test_stats.r2)

```
%pyspark
test_stats = linearModel.evaluate(test_data)
print("RMSE:", test_stats.rootMeanSquaredError)
print("R2:  ", test_stats.r2)
```

```
RMSE: 11.84618875859852
R2:   0.0016427383226924608
```

Took 3 sec. Last updated by anonymous at December 10 2022, 4:10:24 PM.

Until now we have trained the model, but now we must make the prediction for the given scenario. We need to predict what would be the expected score in the first 6 overs if `Mubai Indians` bat against `Kolkata Night Riders`. So, we need to find corresponding feature matrix.

indexed_df.show(5)

```
%pyspark
indexed_df.show(5)

+-------+------------------+------------------+----+------------+------------+
|     id|      batting_team|      bowling_team| sum|batting_coded|bowling_coded|
+-------+------------------+------------------+----+------------+------------+
| 392211|Kolkata Knight Ri...|   Delhi Daredevils|54.0|         2.0|         6.0|
| 419148|Royal Challengers...|Kolkata Knight Ri...|48.0|         1.0|         2.0|
| 501218|     Pune Warriors|    Mumbai Indians|33.0|         9.0|         0.0|
|1082643|     Gujarat Lions| Sunrisers Hyderabad|61.0|        11.0|         7.0|
|1178429| Sunrisers Hyderabad|Royal Challengers...|52.0|         7.0|         1.0|
+-------+------------------+------------------+----+------------+------------+
only showing top 5 rows

Took 1 sec. Last updated by anonymous at December 05 2022, 9:23:00 PM.
```

data_set.show(5)

```
%pyspark
data_set.show(5)

+-------------+-------------+----+----------+
|batting_coded|bowling_coded| sum|  features|
+-------------+-------------+----+----------+
|          2.0|          6.0|54.0| [2.0,6.0]|
|          1.0|          2.0|48.0| [1.0,2.0]|
|          9.0|          0.0|33.0| [9.0,0.0]|
|         11.0|          7.0|61.0|[11.0,7.0]|
|          7.0|          1.0|52.0| [7.0,1.0]|
+-------------+-------------+----+----------+
only showing top 5 rows

Took 0 sec. Last updated by anonymous at December 05 2022, 9:23:03 PM.
```

Now we need to filter out corresponding feature vector for `Mumbai Indians` and `Kolkata Night Riders` Therefore, we inner join **`indexed_df`, and `data_set`** data frames to get the corresponding feature vector.

cond = [indexed_df.batting_coded == data_set.batting_coded, indexed_df.bowling_coded == data_set.bowling_coded]

joinded_df=data_set.join(indexed_df, cond, 'inner')

predict_df=joinded_df[joinded_df.batting_team=='Mumbai Indians'][joinded_df.bowling_team=='Kolkata Knight Riders'].select(['features'])

predict_df.show(1)

```
%pyspark

# Now we need to filter out correspoinding feature vector for `Mumbai Indians` and `Kolkata Night Riders`
# Therefore, we innner join `indexed_df`, and `data_set` dataframes to get the corresponding feature vector
cond = [indexed_df.batting_coded == data_set.batting_coded, indexed_df.bowling_coded == data_set.bowling_coded]
joinded_df=data_set.join(indexed_df, cond, 'inner')
predict_df=joinded_df[joinded_df.batting_team=='Mumbai Indians'][joinded_df.bowling_team=='Kolkata Knight Riders'].select(['features'])
predict_df.show(1)

+---------+
| features|
+---------+
|[0.0,2.0]|
+---------+
only showing top 1 row


Took 2 sec. Last updated by anonymous at December 05 2022, 9:11:07 PM.
```

Predict using the Linear regression model that we have already trained.

prediction = linearModel.transform(predict_df)

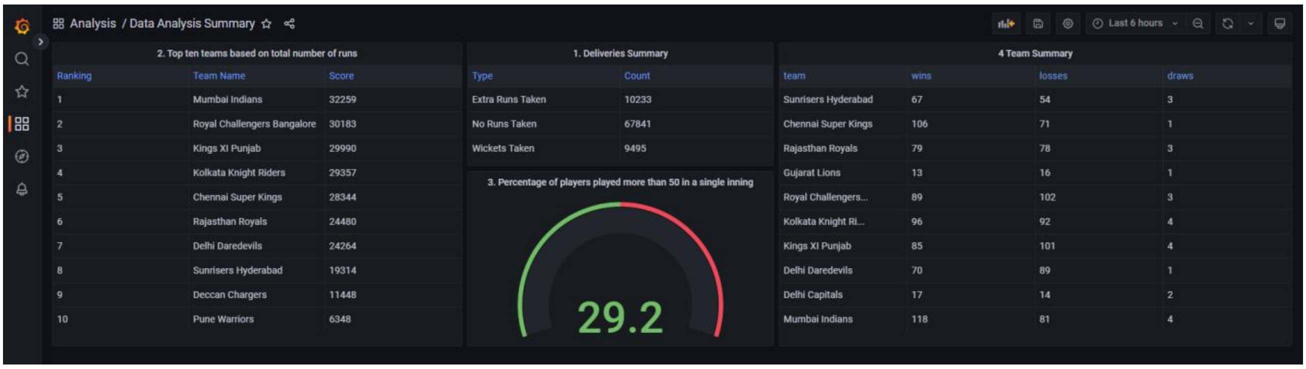
prediction.show(1)

```
%pyspark
prediction = linearModel.transform(predict_df)
prediction.show(1)

+---------+-----------------+
| features|       prediction|
+---------+-----------------+
|[0.0,2.0]|45.44378008259662|
+---------+-----------------+
only showing top 1 row


Took 2 sec. Last updated by anonymous at December 05 2022, 9:11:09 PM.
```

Here we can see the predicted score for `Mumbai Indians` as the batting team, and `Kolkata Knight Riders` as the bowling team is `45.44`

4. Presentation of analysis

For the presentation for the analysis, I used Grafana and Infinity plugin to create a static datastore for the dashboard.



How to run this Dashboard?

In the zip file there is a file called **analysis-dashboard.yaml**, to run this file we need to have Kubernetes cli which is **kubectl** and working cluster. When they are ready, we can simply create the necessary resources using the following command.

kubectl create -f analysis-dashboard.yaml

```
jmadushan@CLLK-JANITHAM:~/rgu/CMM705/cw$ kubectl create -f analysis-dashboard.yaml
configmap/analysis-dashboard created
configmap/grafana-dashboards created
configmap/grafana-infinity created
pod/grafana created
jmadushan@CLLK-JANITHAM:~/rgu/CMM705/cw$
```

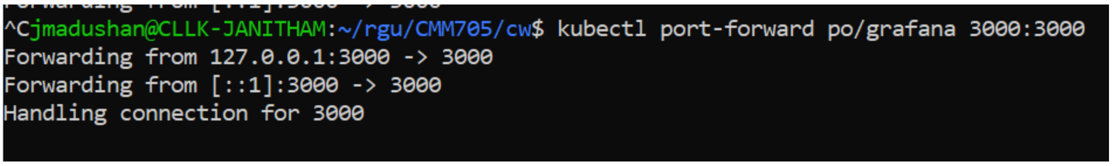
Let me describe briefly about **analysis-dashboard.yaml** file, it contains following **Kubernetes** resources with corresponding usages.

Resource	**Usage**
ConfigMap (analysis-dashboard)	Contains dashboard json file which is the json representation of our dashboard
ConfigMap (Grafana-dashboards)	Dashboard provisioners to automate provisioning
ConfigMap (Grafana-infinity)	Data source provisioners to automate infinity data source provisioning
Pod (Grafana)	Grafana dashboard pod containing the Grafana container

And this is a fully automated static dashboard which will automatically provisioning the resources that is necessary. **Configuration as code (CAC)** paradigm has been used here.

And to access this dashboard we need to create a bonded port with local host. For that we could use the following command to open a port locally.

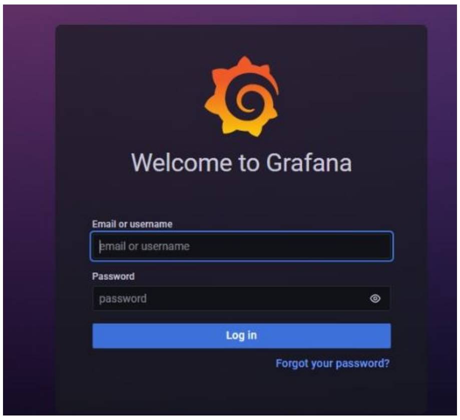
kubectl port-forward po/grafana 3000:3000



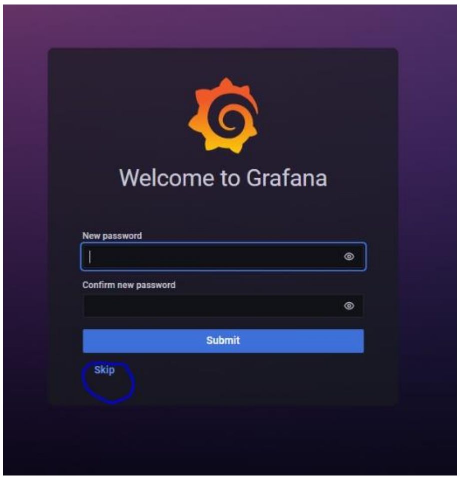
To access this dashboard, we could simply use a web browser with localhost using the following URL.

http://localhost:3000

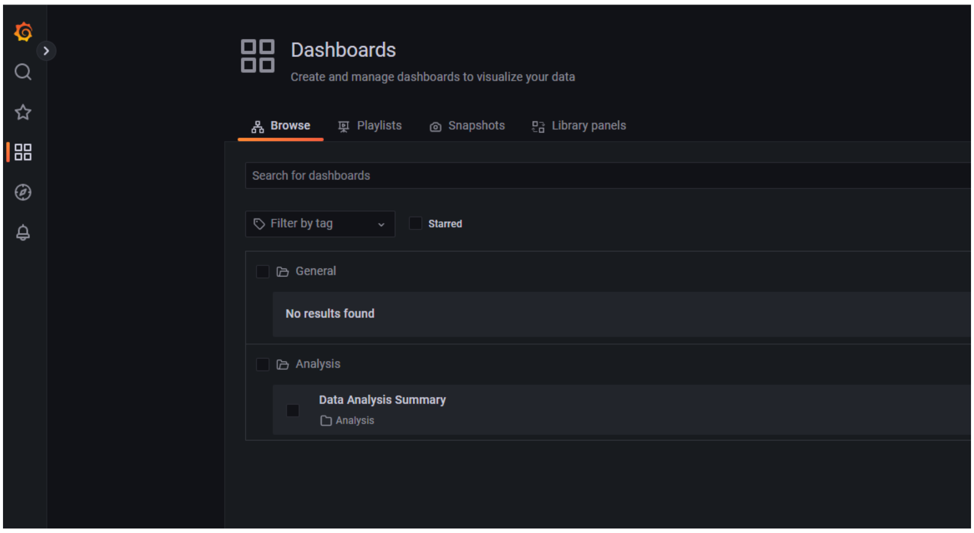
then we will get a UI like this



Default username password would be **admin:admin**



Then skip the resetting password, and next we can browse the dashboard as follows.



References

https://masum-math8065.medium.com/how-to-convert-categorical-data-into-numeric-in-pyspark-2202407f5fac

https://blog.devgenius.io/apache-storm-on-kubernetes-b0cda6b39546

https://www.elastic.co/guide/en/elasticsearch/hadoop/current/mapreduce.html

https://cazton.com/consulting/enterprise/lambda-architecture

https://www.elastic.co/blog/building-real-time-dashboard-applications-with-apache-flink-elasticsearch-and-kibana

https://www.elastic.co/what-is/elasticsearch