Coursework Title: Big Data Analytics: Group Coursework

Module Name: Big Data Analytic

Module Code: CN7022

Tutor: Amin Karami

Group 4:

- **John Olorunuyi**
- **Chung Hoi Chiu**

Submission date: 2nd January 2018

Table Of Contents

Introduction

Big data analytics can be defined as the method of analzing big data. Big data on the other can be described as a very large data (structure and unstructure data) that traditional database cannot handle. This big data is gathered from variety of sources like sensors, sales records, video, images and social networks (twitter, facebook, pininterest and google). The main purpose of big data analytics is to discover hidden patterns and connections that mostly not visible but can provide great business decisions. Conventional data warehousing software are not well suited for today's type of data like unstructure and high processing requirements, therefore newer technologies like Apache Hadoop, MapReduce or NoSQL are now used to analysed big data ov er clustered systems.

Apache hadoop is an open source software which can be used for both distributed storage and processing of large data. The storage part of hadoop is known as hadoop distributed file system (HDFS) and the processing part is called the mapReduce.

Hadoop comes with collections of additional module or software (ecosystem) as shown below, that can be installed on top or alongside like Pig, HBase, Hive, spark and so on.

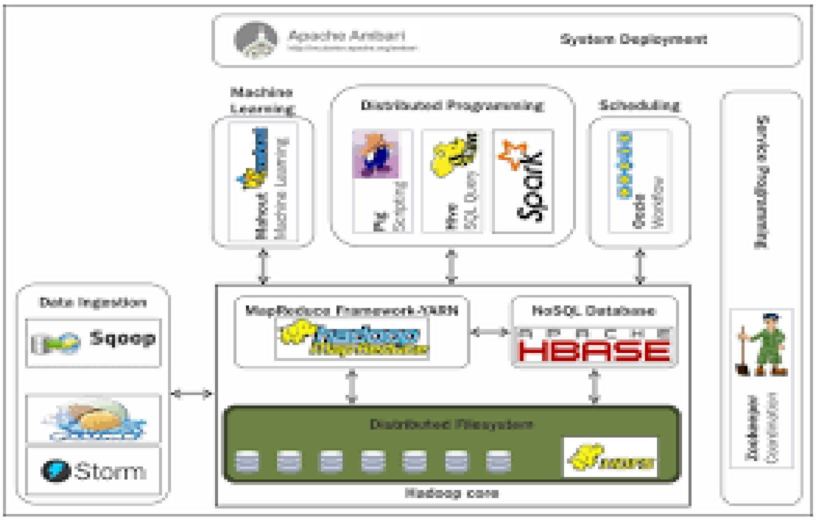


Figure 1: Hadoop Eco-system (O'Reilly | Safari, 2018)

Hadoop can run as a single node or multi-node, for the this coursework, we will be running multi-node (2 nodes) by running two hadoop daemons indifferent virtual machine on a single machine. The two nodes will be managed by apache ambari.

Apache Ambari

Apache ambari is an open source software designed to provide, manage and monitor apache haadoop clusters. Ambari provides through its RESTful APIs an easy to use web based hadoop management system.With ambari, system administartor can easily install hadoop services across clusters, manage cluster by stopping, starting and configuration and also use povided dashboard to manage health and status of clusters.

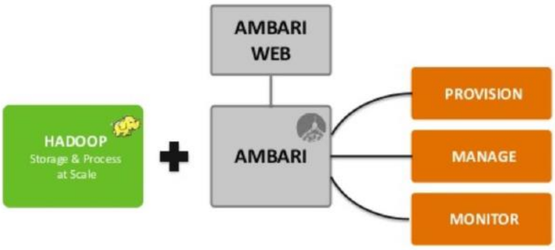


Figure 2: Apache Ambari Block Diagram (Dr Amin, Big Data Tutorial. UEL 2017)

There are two main components in Apache Ambari architecture nameky: Ambari Server and Ambari Agent.

- Ambari Server deals with interaction of all agents installed on the nodes
- Ambari Agent with the help of various operational metrics deals with the update of all nodes.
  Ambari Server and Ambari Agent block diagram is shown below:

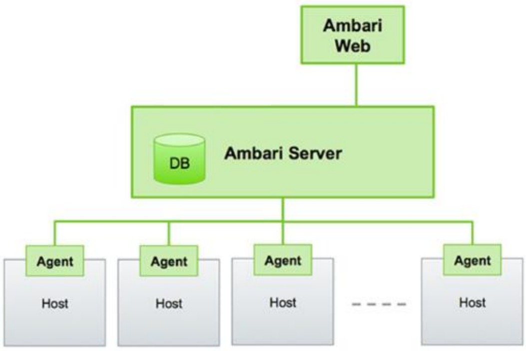


Figure 3: Ambari Server and Ambari Agent Block Diagram (Dr Amin, Big Data Tutorial. UEL 2017)

For this coursework, the big data stack installation will follow the above block diagram:

- Ambari Web (Apache http server) will run ambari web user interface (UI)
- Node1.group4.com setup as tha Ambari Server
- Node2.group4.com set as Agent and password-less.

Big Stack Installation Process

The environment for installing our Big stack is shown below:

- Operating system: Two Ubuntu 14.04
- Vendor: Hortonworks
- Ambari: 2.5.0
- HDP: 2.6
- Setup: Multinode Cluster (2 nodes)
  node1.group4.com
  node2.group4.com

  Two Ubuntu operating system were setup on virtual machime named: node 1 and node2 and root access configuration as shown in the figure below:

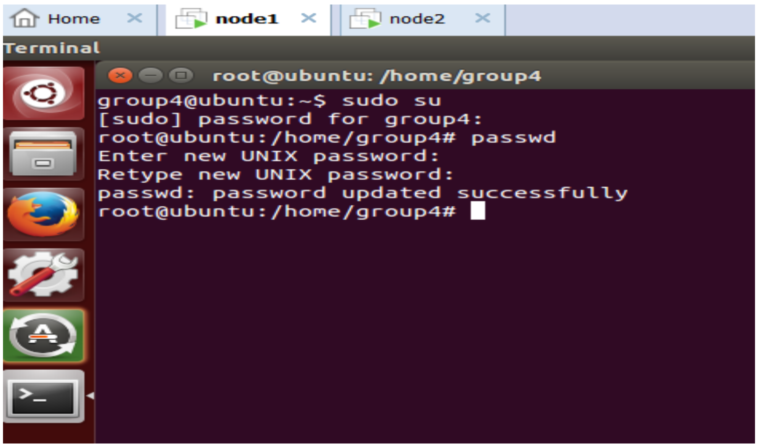


Figure 4: Node1 and Node2 setup

SSH Server installed on both nodes using the following command:

$ apt-get install openssh-server

The ssh_config file was then open with command $gedit /etc/sshd_config, where the following were changed:

#PasswordAuthentication yes was found and the # was removed. Also PermitRootLogin without-password was also found and the without-password was removed and replace with yes. The ssh was restart with command $ service ssh restart.

2.4

Firewall and iptables were disable with command:

$ ufw disable

$ iptables –F

2.5

Hostname and FQDN check with the command:

$hostname

$hostname –f

The results of both commands should be the same as shown in the figure below. Hostname shows the system hostname (local addressing) whereas FQDN shows short hostname and DNS domain name. to make the hostname and FQDN are the same by setting up a new hostname for both nodes as

- node1.group4.com
- node2.group4.com

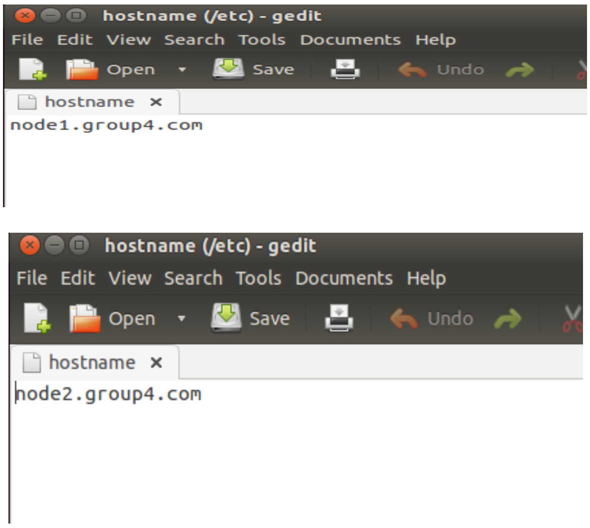


Figure 5: Rename Hosts

To avoid DNS issues, each node unique FQDN was added into each host with the command:

(Ifconfig was used to find out IP address in each machine)

$gedit /etc/hosts

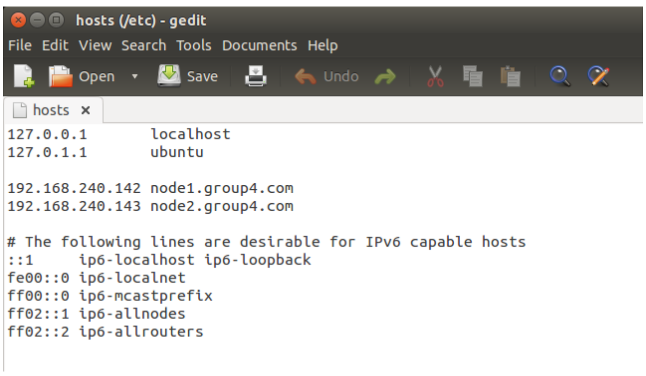


Figure 6: Unique FQDN added into each host

The hostname service was then restart as shown below:

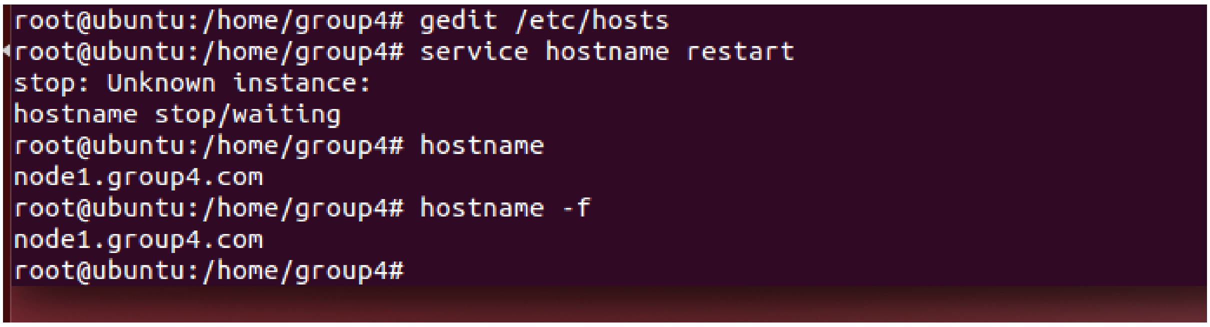


Figure 7: Node1 Hostname Restart

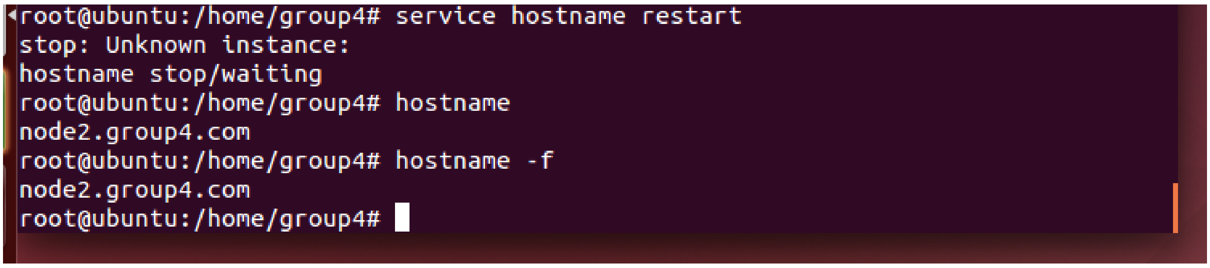


Figure 8: Node2 Hostname Restart

2.6

Enable NTP

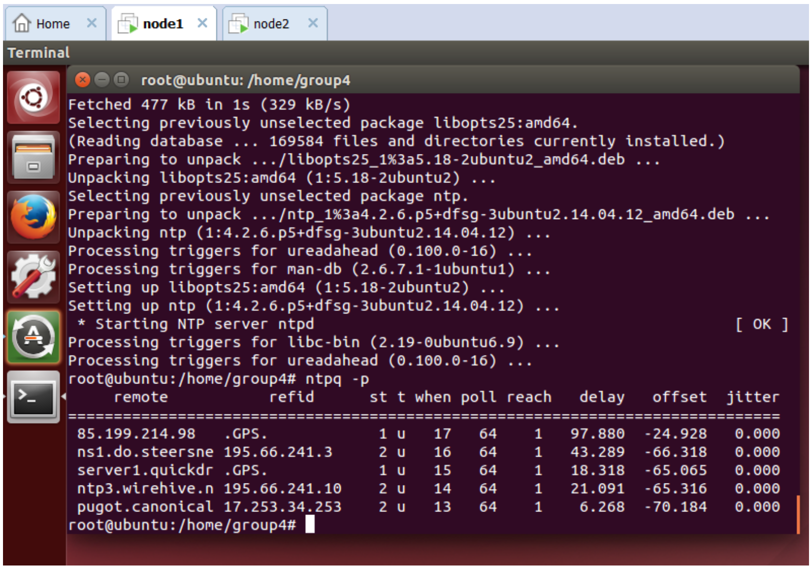


Figure 9: Node1 Enable NTP and confirm

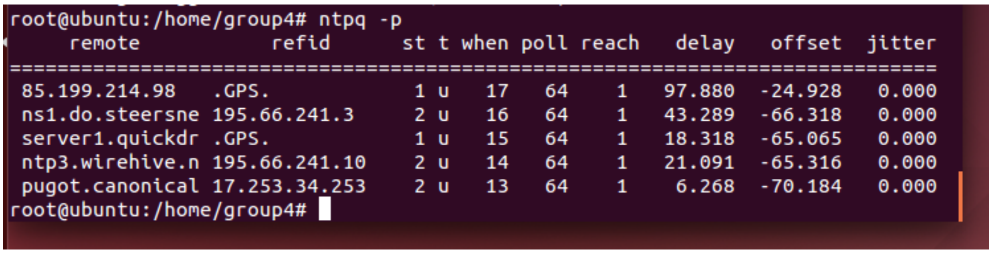


Figure 10: Node2 Enable NTP and confirm

2.7

Password-Less Setup (Server Only)

To allow Ambari server to automatically install ambary on all agents on cluster's hosts, password-less SSH must be setup between the server host and the second host in the cluster. To do this Ambari server host uses SSH public key authentication to remotely access and install the agent. This was done with the command:

$ ssh-keygen

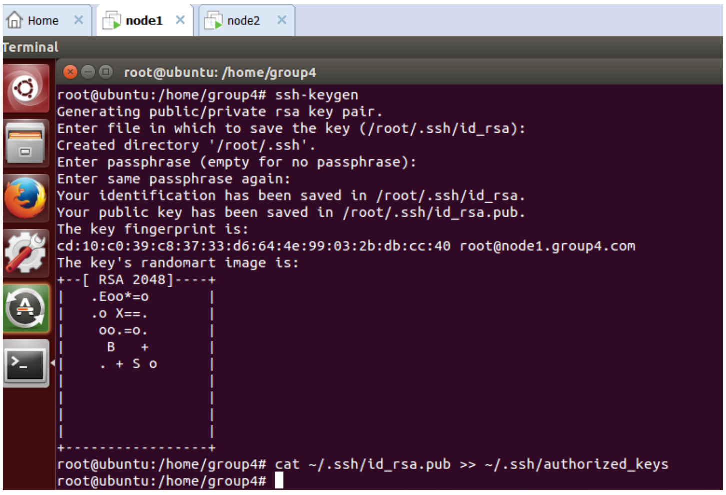


Figure 11: SSH Key-gen

The SSH public was the copied into authorized_keys variable with command:

$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys

Also the permission was also set on the .ssh directory and authorized_keys with command:

$ chmod 700 ~/.ssh/

A new folder called the .ssh was created in node 2 for authorized_keys as shown below:

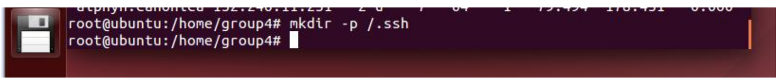


Figure 12: Create new folder in node

The generated was then copied into the folder .ssh in node2, this was done for password-less use of ssh by Ambari Server.

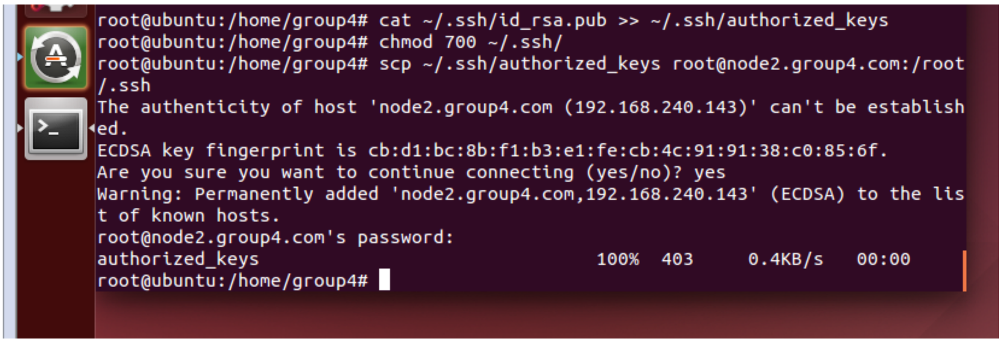


Figure 13: copy key from server to node2

Password-less connection was tried from server to node2 using command ssh and no password was asked (successful)

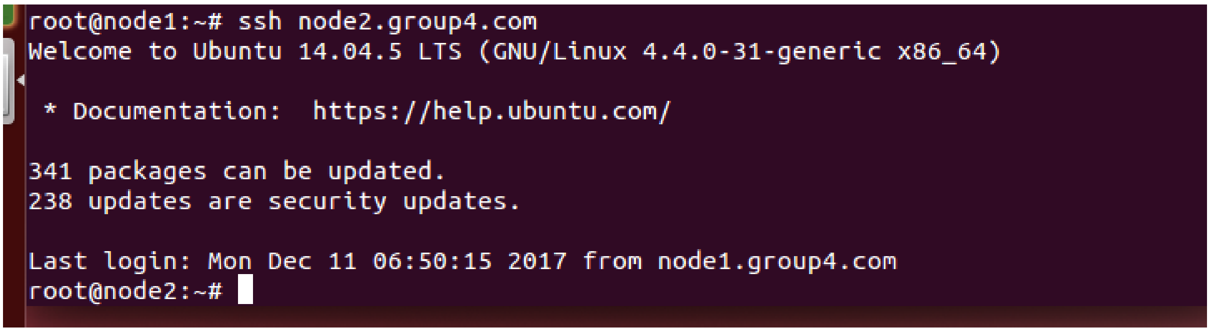


Figure 14: Password-less connection the server to node2

3.0

Apache HTTP Server Installation (Server)

In order to run web-based Ambari service, we need to install Apache HTTP server with the command:

$ apt-get install apache2

And then start the Ambari service with the command:

$ service apache2 start

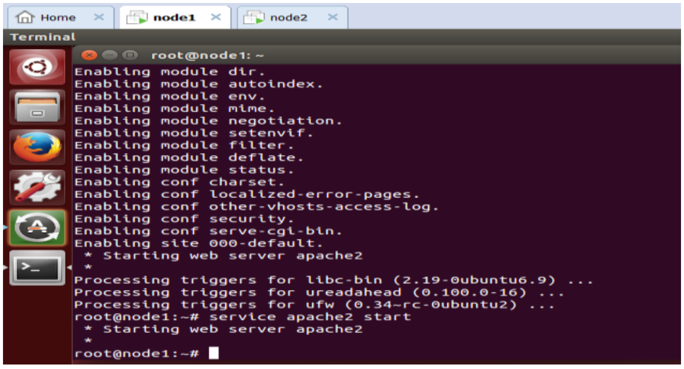


Figure 15: Apache2 server start confirmation

3.1

Ambari Repository Download (Server)

The Ambari repo files were downloaded from hortonworks to installation host using below command:

$ wget -O /etc/apt/sources.list.d/lloca.list http://public-repo-1.hortonworks.com/ambari/ubuntu14/2.x/updates/2.5.0.3/ambari.list

And also the recv keys with command:

$ apt-key adv –recv-keys –keyserver hkp://keyserver.ubuntu.com:80 B9733A7A07513CAD

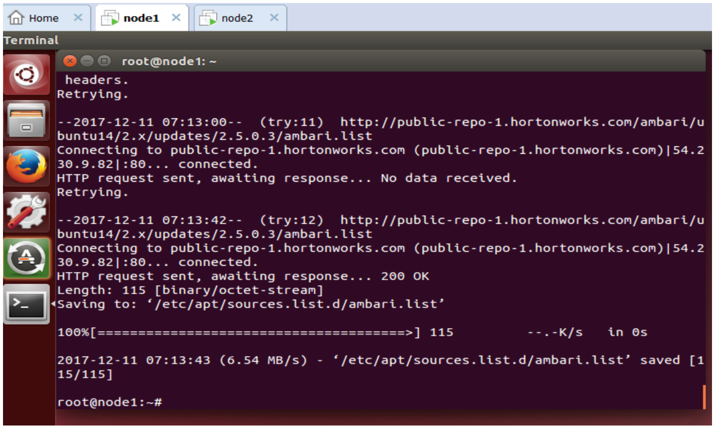


Figure 16: Password-less connection the server to node2

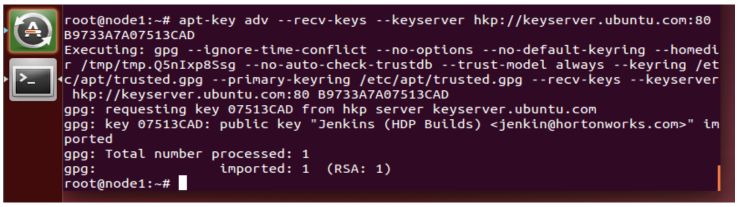


Figure 17: Ambari repo files downloaded

An update was performed and then ambary packages were confirmed that they were downloaded successfully with the following command:

$ apt-cache showpkg lloca-server

$ apt-cache showpkg lloca-agent

$ apt-cache showpkg lloca-metrics-assembly

3.2

Ambari Server Setup

The setup is necessary before starting local Server to enable local to communicate with database, install jdk and to customize the user account. The Ambari Server daemon is going to run on this user account. This was done with following command:

$ apt-get install local-server

The Ambari Server setup was completed successfully as shown in screen shot below:

```
..........
Adjusting ambari-server permissions and ownership...
Ambari Server 'setup' completed successfully.
root@node1:~#
```

Figure 18: Ambari Server setup completed

The lloca Server was started successfully the Server started listening on port 8080 as shown below:

```
root@node1:~# ambari-server start
Using python  /usr/bin/python
Starting ambari-server
Ambari Server running with administrator privileges.
Organizing resource files at /var/lib/ambari-server/resources...
Ambari database consistency check started...
Server PID at: /var/run/ambari-server/ambari-server.pid
Server out at: /var/log/ambari-server/ambari-server.out
Server log at: /var/log/ambari-server/ambari-server.log
Waiting for server start....................................
Server started listening on 8080

DB configs consistency check: no errors and warnings were found.
Ambari Server 'start' completed successfully.
root@node1:~#
```

Figure 19: Ambari Server started

3.3

Configuring and Deploying Hadoop Cluster

Since the Ambari Server is now running, next is log to Ambari web browser using the following address http://localhost:8080, using admin as both user name and password. A new cluster was created called the BigData. To select version, the following were selected

- Select the default HDP 2.6.
- Use public repository
- Remove unnecessary Operating System and keep Ubuntu 14.

To install option, enter the list of all host into target hosts box, provide the SSH private key, made SSH User Account as root and SSH port as 22.

3.4

Confirming hosts was done to confirm that Ambari located the hosts for the cluster and to check they have correct directories, packages and processes to continue the installation. This was done successfully as shown by the figure below:

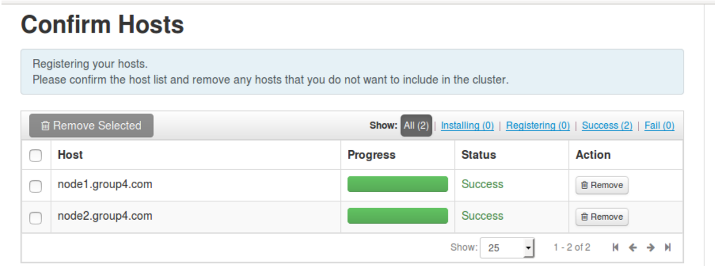


Figure 21: Confirmed Hosts

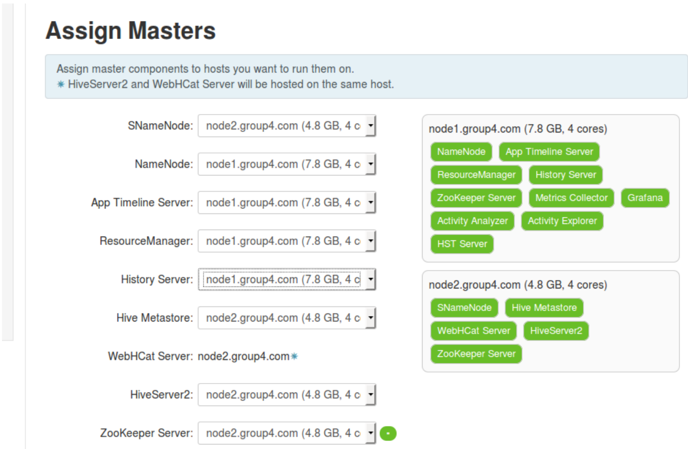


Figure 22: Assign Masters

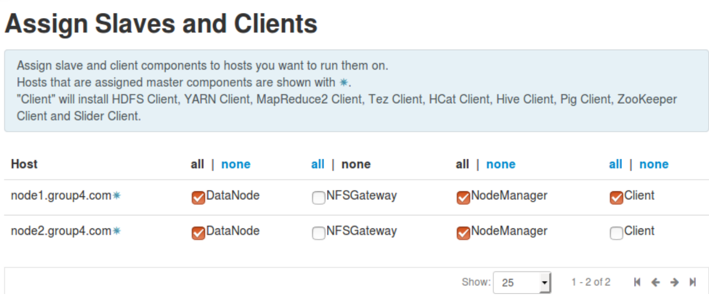


Figure 23: Assign Slaves and Clients

Installation, start and test were successful as shown in the figure below:

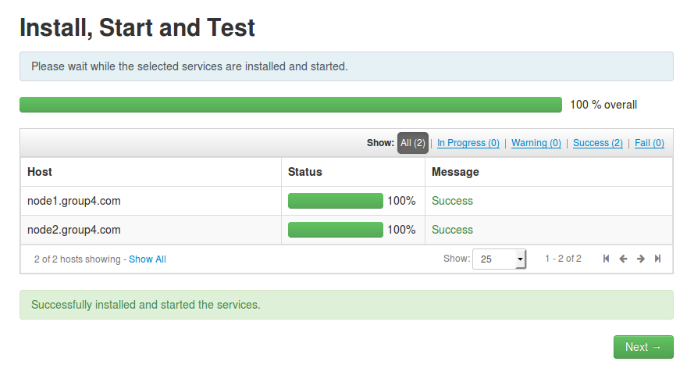


Figure 24: Successful Install, Start and Test

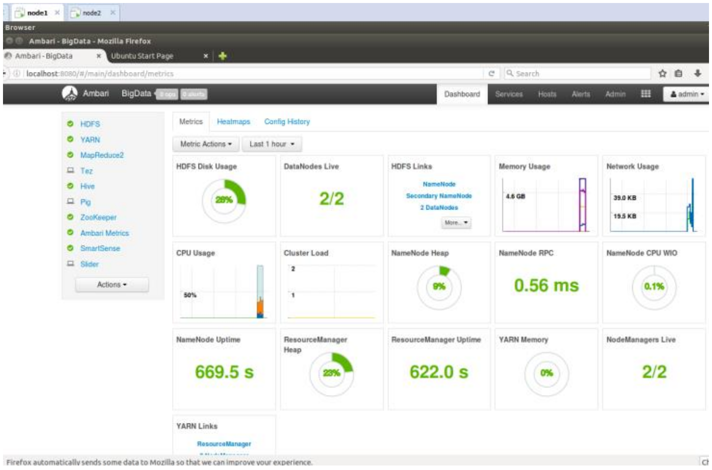


Figure 25: Apache Ambari view

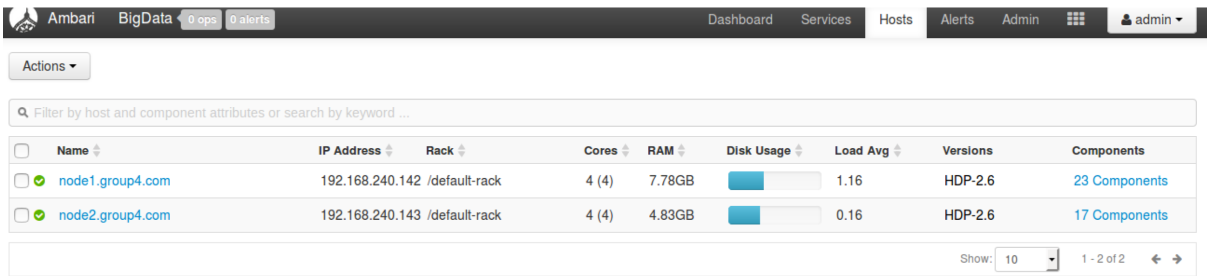


Figure 26: Apache Ambari hosts view

To work with HDFS, we created directory for user as cluster administrator with the following commands:

$ su – hdfs // connect to HDFS as hdfs user

$ hdfs dfs –mkdir /user/admin

$ hdfs dfs –chown admin:lloca /userS/admin

$ hdfs dfs –chmod 777 /user/admin

$ exit

4.1 Hive Querying

For hive querying we used two datasets Salaries.csv and Batting.csv already loaded into the HDFS.

Task a:

Two data set were (Salaries.csv and battingn.csv) were downloaded and were loaded into HDFS using two methods.

First method was by using hdfs commands as shown below:

$ hdfs dfs -put /home/group4/Downloads/Salaries.csv /user/admin

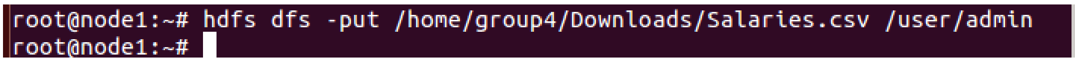


Figure 26: Upload Salaries.csv into HDFS

Second method used was to using upload link in Ambari Web. Two files are shown in user/admin folder as shown below:

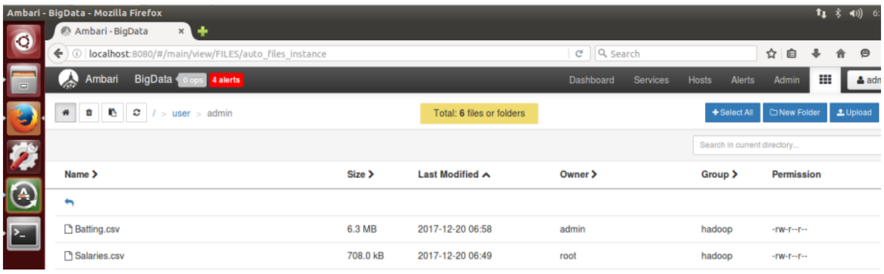


Figure 27: Uploaded Files in HDFS

Task b: Print the names of all teams that every player had a salary higher than 500,000$ in the year of 2000.

Firstly, a table called tempSalaries was created as shown below:

$ create table tempSalaries (col_value STRING)

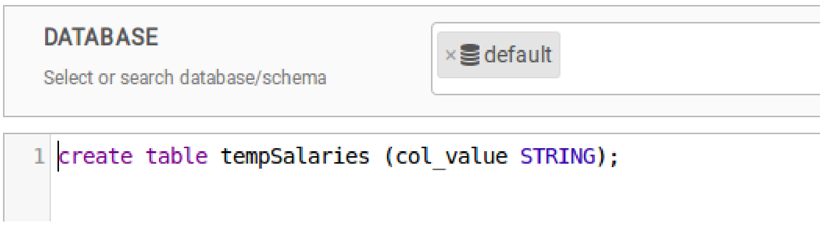


Figure 28: Create table tempSalaries

Load the data file Salaries.csv into the table tempSalaries

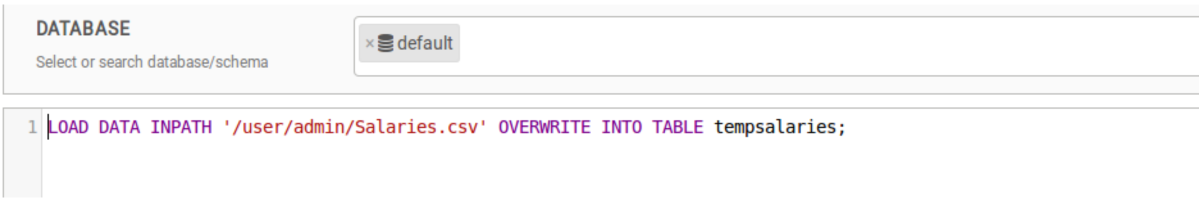
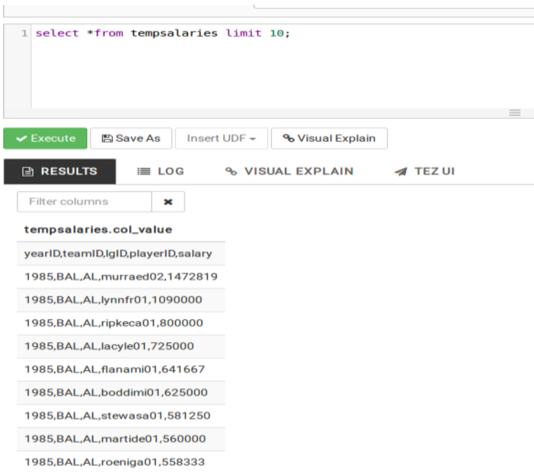


Figure 29: Load the data file

After loading the data, we checked to make sure the tempSalaries was populated with data from Salaries.csv.

$ select *from tempSalaries limit 10;



\

Figure 29: Check the data file

Table salaries was created with five colown

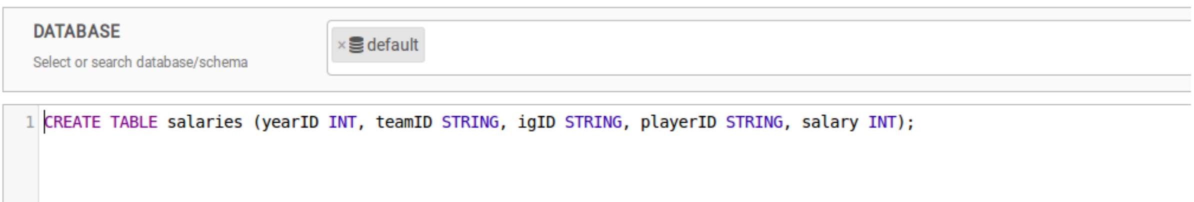


Figure 30: Create table salaries

Populate table salaries with data from tempSalaries by using **regexp pattern**:

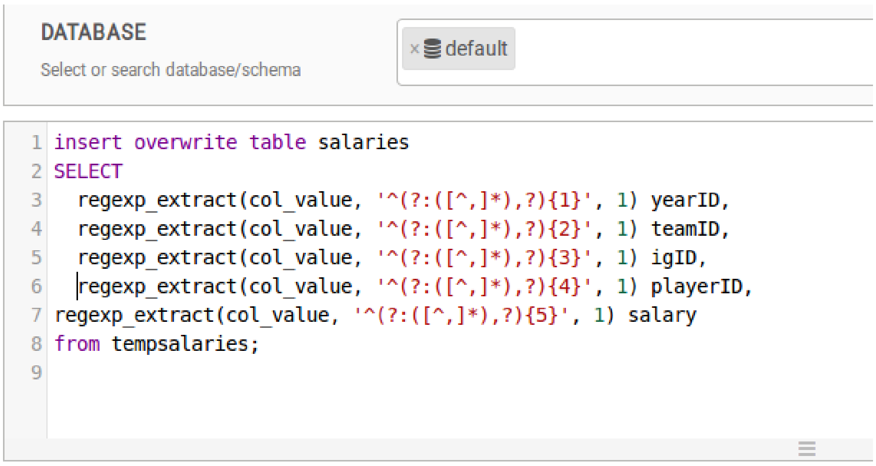


Figure 31: Populate table salaries

Task 1B

Print the names of all teams that every player had a salary higher than 500,000$ in the year of 2000.

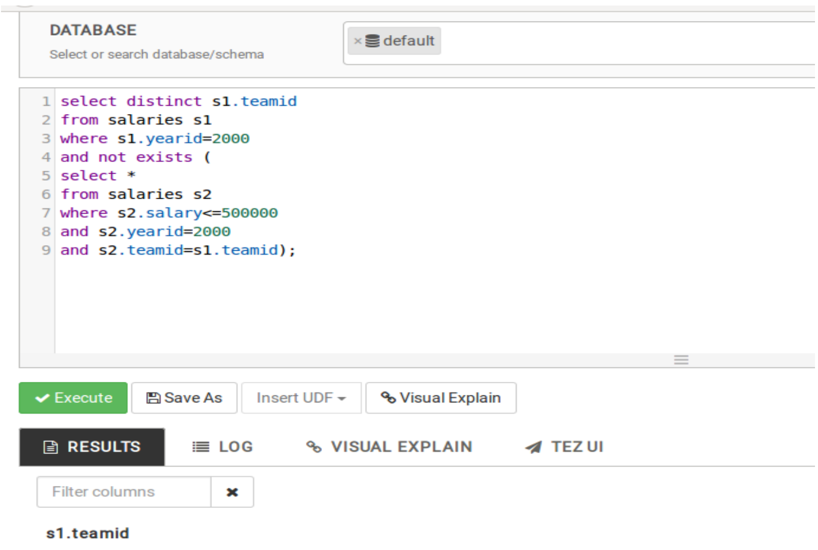


Figure 32: Task 1 screen shot

No output for salary higher than $500.000 but If salary is reduced to $200,000, then we get output.

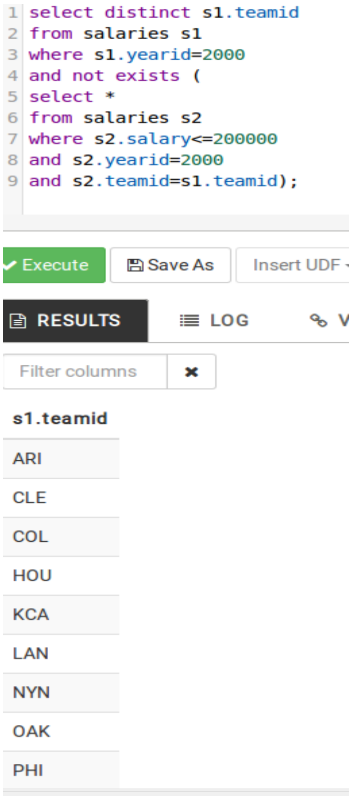


Figure 33: Task 1 screen shot (Salaries reduced)

C. Print all the teams with their corresponding average salary in 1988.

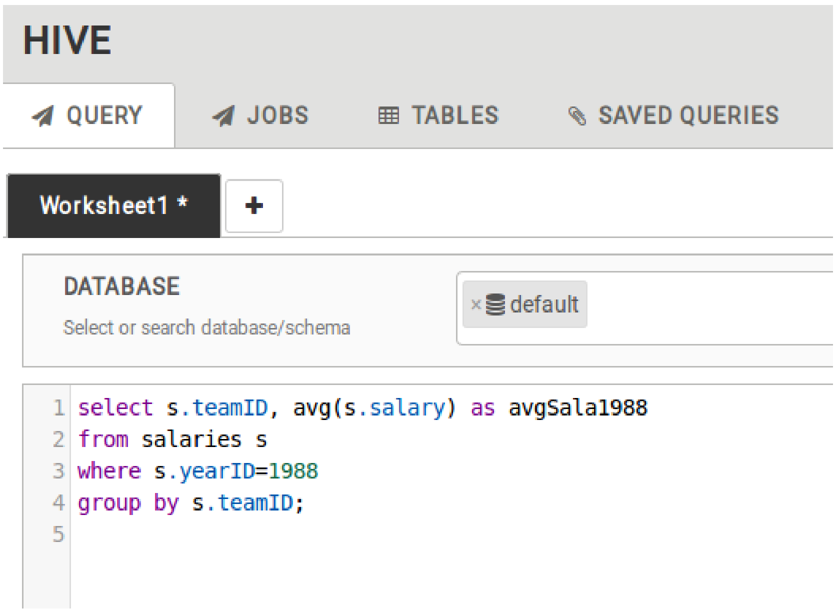


Figure 34: Task 2 screen shot

s.teamid	avgsala1988
ATL	438902.5517241379
BAL	501187.962962963
BOS	579003.8333333334
CAL	426692.4285714286
CHA	266250.0
CHN	524767.92
CIN	355536.36
CLE	406204.54545454547
DET	559546.5652173914
HOU	472544.8846153846
KCA	559867.7692307692
LAN	561683.8333333334
MIN	541855.0434782609
ML4	381909.0909090909
MON	384133.32
NYA	648038.4
NYN	610772.56
OAK	421304.347826087
PHI	532230.7692307692
PIT	222166.66666666666
SDN	354111.18518518517
SEA	282401.92307692306
SFN	495200.0
SLN	477037.037037037
TEX	242824.13636363635
TOR	470816.3461538461

Figure 35: Task 2 result

DATABASE

Select or search database/schema

× default

```
create table tempbatting (col_value String)
```

Figure 36: Create table tempbatting

DATABASE

Select or search database/schema

× default

```
LOAD DATA INPATH '/user/admin/Batting.csv' OVERWRITE INTO TABLE tempbatting;
```

Figure 37: Load data into table tempbatting

DATABASE

Select or search database/schema

× default

```
create table batting (playerID String, yearID int,
stint int, teamID String, IgID String, G int, G_batting int, AB int, R int, H int, n2B int,
n3B int, HR int);
```

Figure 38: Create table batting

DATABASE

Select or search database/schema

× default

```
insert overwrite table batting
SELECT
regexp_extract(col_value, '^(?:([^,]*),?){1}', 1) playerID,
regexp_extract(col_value, '^(?:([^,]*),?){2}', 1) yearID,
regexp_extract(col_value, '^(?:([^,]*),?){3}', 1) stint,
regexp_extract(col_value, '^(?:([^,]*),?){4}', 1) teamID,
regexp_extract(col_value, '^(?:([^,]*),?){5}', 1) IgID,
regexp_extract(col_value, '^(?:([^,]*),?){6}', 1) G,
regexp_extract(col_value, '^(?:([^,]*),?){7}', 1) G_batting,
regexp_extract(col_value, '^(?:([^,]*),?){8}', 1) AB,
regexp_extract(col_value, '^(?:([^,]*),?){9}', 1) R,
regexp_extract(col_value, '^(?:([^,]*),?){10}', 1) H,
regexp_extract(col_value, '^(?:([^,]*),?){11}', 1) n2B,
regexp_extract(col_value, '^(?:([^,]*),?){12}', 1) n3B,
regexp_extract(col_value, '^(?:([^,]*),?){13}', 1) HR
from tempbatting;
```

Figure 39: Populate table batting

Task d. Print the 15 players who made the most hit in 1998. The display must be the PlayerID and the amount of hits.

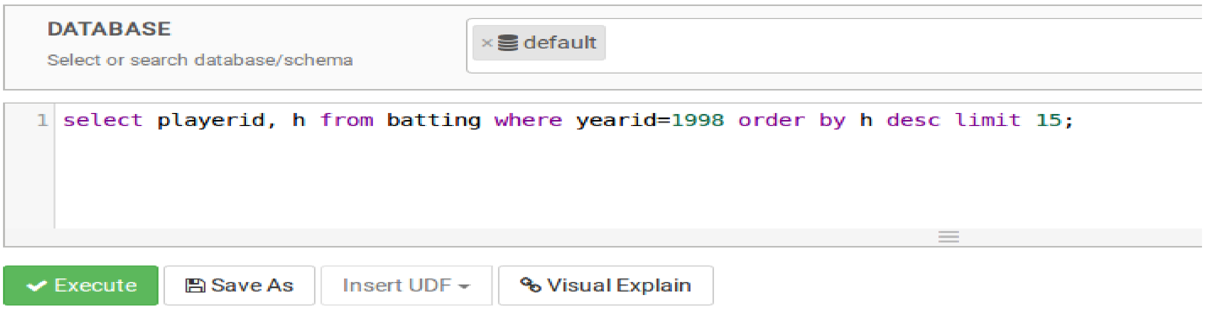
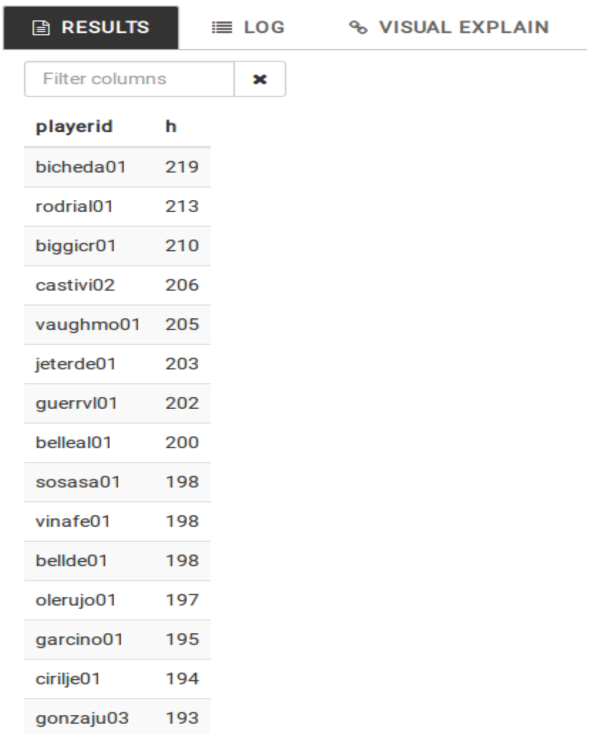


Figure 40: Task 3



playerid	h
bicheda01	219
rodrial01	213
biggicr01	210
castivi02	206
vaughmo01	205
jeterde01	203
guerrvl01	202
belleal01	200
sosasa01	198
vinafe01	198
bellde01	198
olerujo01	197
garcino01	195
ciriIje01	194
gonzaju03	193

Figure 41: Task 3 result

Task e. Print the player in the ML1 team with the most runs in 1960. The display must be the PlayerID and the number of runs.

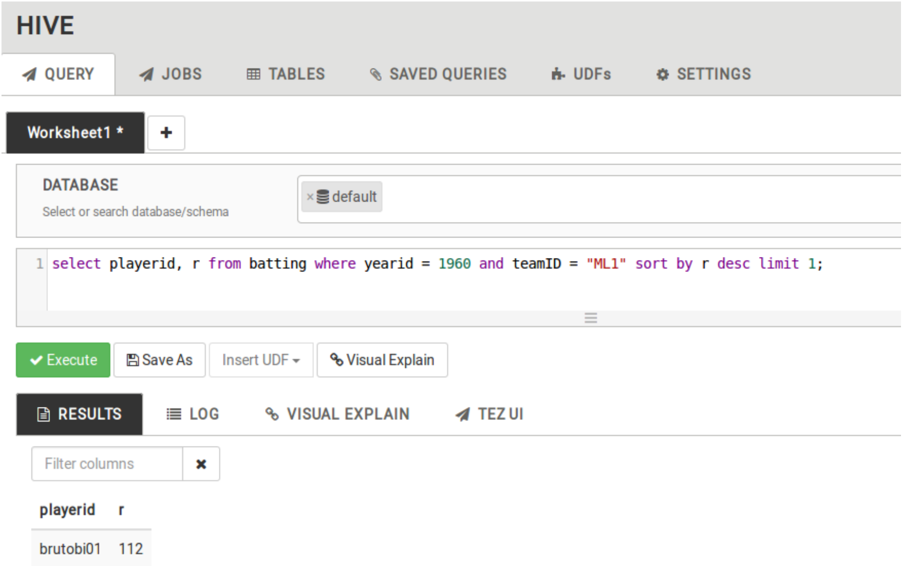


Figure 42: Task 4

Task f.

Print those players with a salary above 500,000$ in 2002 who have made more than 30 homeruns. Display the players, their number of homeruns and salary.

```
SELECT salaries.playerid, batting.hr, salaries.salary
FROM salaries
INNER JOIN batting
ON salaries.playerid=batting.playerid
AND salaries.teamid=batting.teamid
AND salaries.igid=batting.igid
AND salaries.yearid=batting.yearid
WHERE salaries.yearid=2002
AND batting.hr>30
AND salaries.salary>500000
ORDER BY salaries.playerid;
```

Figure 43: Task 5

Results

salaries.playerid	batting.hr	salaries.salary
bagweje01	31	11000000
batisto01	31	4900000
bondsba01	46	15000000
burksel01	32	6666667
burrepa01	37	1905000
chaveer01	34	2125000
delgaca01	33	19400000
giambja01	41	10428571
gilesbr02	38	8063003
greensh01	42	13416667
guerrvl01	39	8000000
jonesan01	35	10000000
kentje01	37	6000000
ordonma01	38	6500000
palmera01	43	8712986
piazzmi01	33	10571429
pujolal01	34	600000
ramirma02	33	15462727
rodrial01	57	22000000
soriaal01	39	630000
sosasa01	49	15000000
tejadmi01	34	3625000
thomeji01	52	8000000

Figure 44: Task 5 result

4.2

Zeppelin

Apache Zeppelin is an open source software for data visualization which is web based and enables interactive data analytics. Zeppelin can be installed as a standalone tool or can be used from Apache Ambari.

For this course, a standalone zeppelin was use for visualization.

To install zeppelin, the following steps were followed:

Installed MYSQL with command:

$ apt-get install mysql-server

(No password required)

Install the MySQL Java Connector

$ apt-get install libmysql-java

created soft link for connector in Hive lib directory with command below (HIVE_HOME = /usr/local/hive):

$ ln -s /usr/share/java/mysql-connector-java.jar /user/local/hive/lib/mysql-connector-java.jar

Figure 45: Zeppelin setup

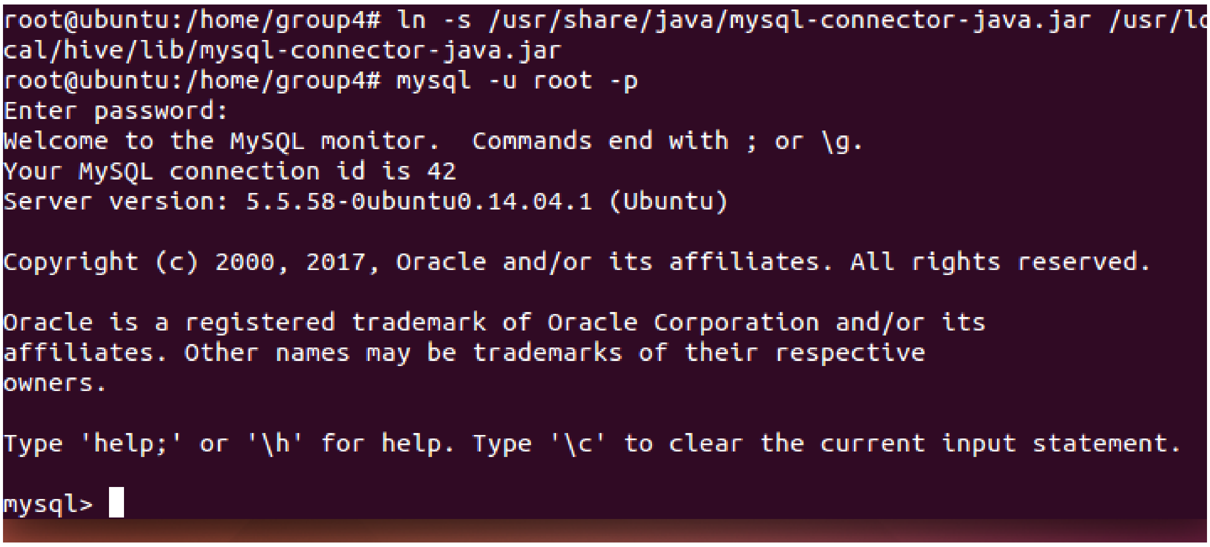
Initial database was create using hive-schema-2.1.0.mysql.sql file with commands:

$ mysql -u root -p

mysql> CREATE DATABASE metastore;

mysql> USE metastore

mysql> SOURCE $HIVE_HOME/scripts/metastore/upgrade/mysql/hiveschema-2.1.0.mysql.sql;



```
root@ubuntu:/home/group4# ln -s /usr/share/java/mysql-connector-java.jar /usr/lo
cal/hive/lib/mysql-connector-java.jar
root@ubuntu:/home/group4# mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 42
Server version: 5.5.58-0ubuntu0.14.04.1 (Ubuntu)

Copyright (c) 2000, 2017, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Figure 46: MYSQL setup

MySQL User account was created and the file hive-site.xml were modified and core-site.xml in Hadoop was configured the apache was downloaded and then move to folder /usr/local/zeppelin

zeppelin-env.sh was configured and port number in zeppelin-site.xml from 8080 to 8090

Also copied Hive JDBC jar file in Zeppelin JDBC interpreter directory in order for hive to ececute query with jdbc

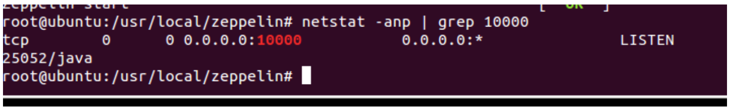
$ cp /usr/local/hive/lib/hive-jdbc-2.1.0.jar /usr/local/zeppelin/interpreter/jdbc/

To start working with zeppelin, start Hadoop, change to home directory of zeppelin and the start zeppelin:

```
root@ubuntu:/usr/local/zeppelin# start-all.sh
This script is Deprecated. Instead use start-dfs.sh and start-yarn.sh
17/12/27 11:27:35 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
Starting namenodes on [localhost]
localhost: namenode running as process 13438. Stop it first.
localhost: datanode running as process 13596. Stop it first.
Starting secondary namenodes [0.0.0.0]
0.0.0.0: secondarynamenode running as process 13792. Stop it first.
17/12/27 11:27:39 WARN util.NativeCodeLoader: Unable to load native-hadoop libra
ry for your platform... using builtin-java classes where applicable
starting yarn daemons
resourcemanager running as process 13951. Stop it first.
localhost: nodemanager running as process 14081. Stop it first.
root@ubuntu:/usr/local/zeppelin# cd /usr/local/zeppelin
root@ubuntu:/usr/local/zeppelin# bin/zeppelin-daemon.sh start
Pid dir doesn't exist, create /usr/local/zeppelin/run
Zeppelin start                                             [  OK  ]
root@ubuntu:/usr/local/zeppelin#
```

Figure 46: Start Zeppelin

Start Hive Server2



```
root@ubuntu:/usr/local/zeppelin# netstat -anp | grep 10000
tcp        0      0 0.0.0.0:10000           0.0.0.0:*               LISTEN
25052/java
root@ubuntu:/usr/local/zeppelin#
```

Figure 47: Start Hive Server

Before working with zeppelin, the two datasets were stored in HDFS

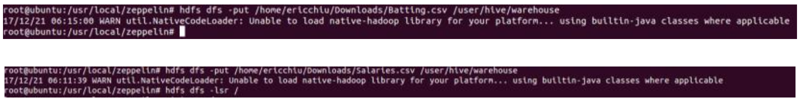


Figure 48: Storing the dataset into HDFS

Interpreter in zeppelin was created selecting interpreter from anonymous user, looked for jdbc and the following changes were done

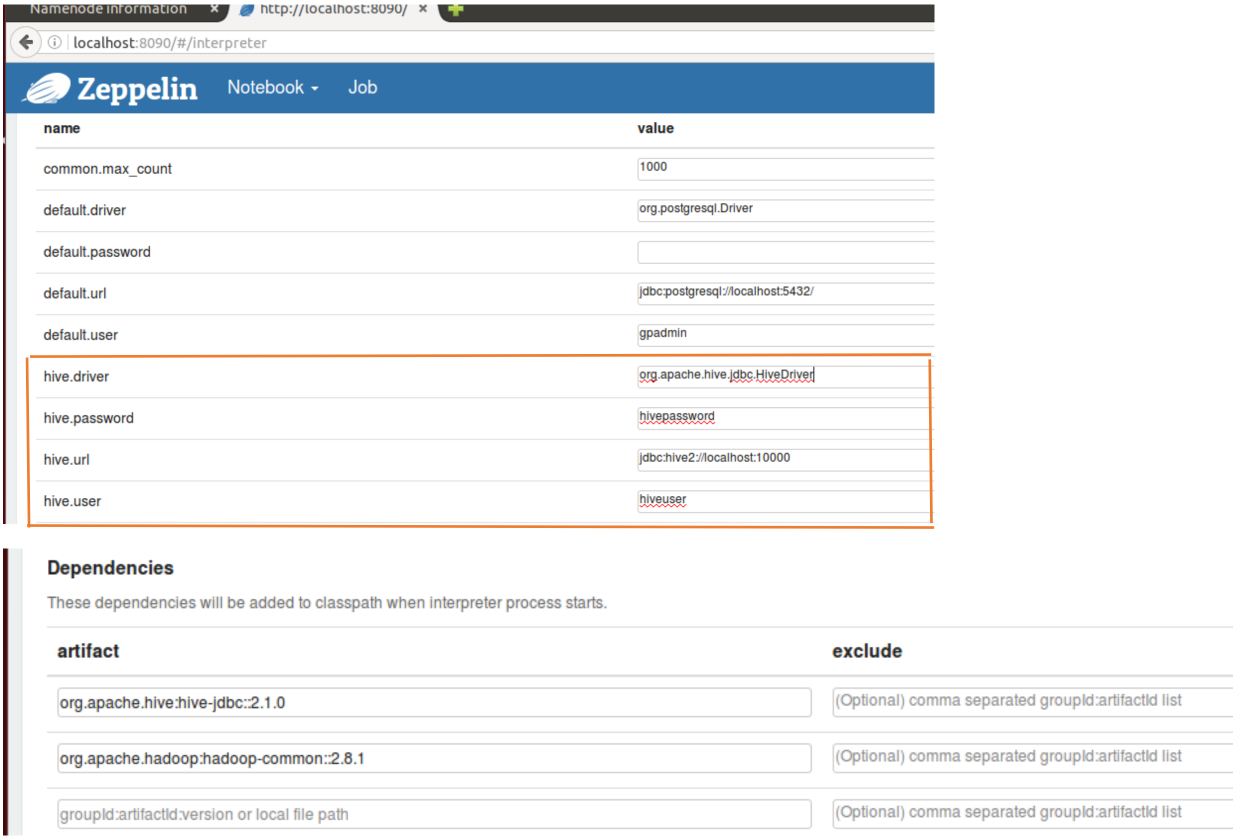


Figure 49: Edit jdbc configuration

Changes were then saved, the waited for jdbc updated with green colour.

A new notebook called group4 was created with jdbc as interpreter

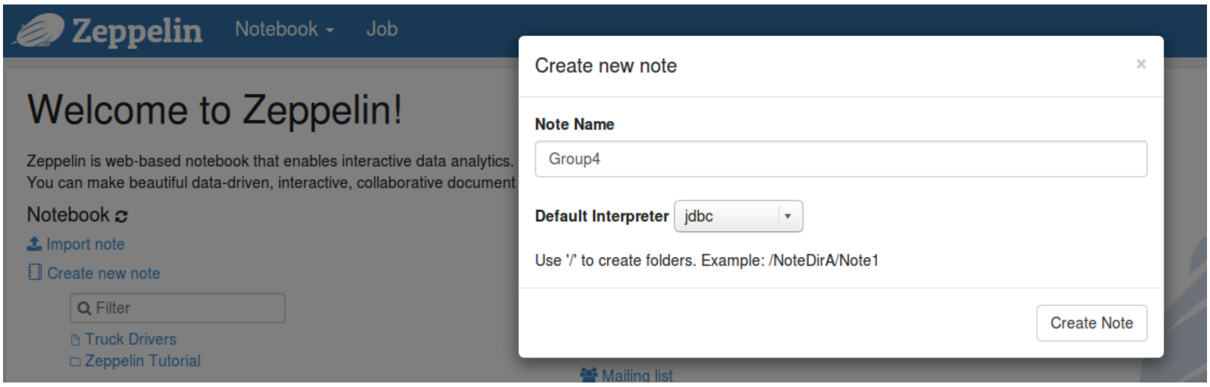


Figure 50: New note created

4.3

Zeppelin Querying

Create table tempSalaries (col_value STRING);

Load DATA INPATH '/user/hive/warehouse/Salaries.cvs' OVERWRITE INTO TABLE tempSalaries;

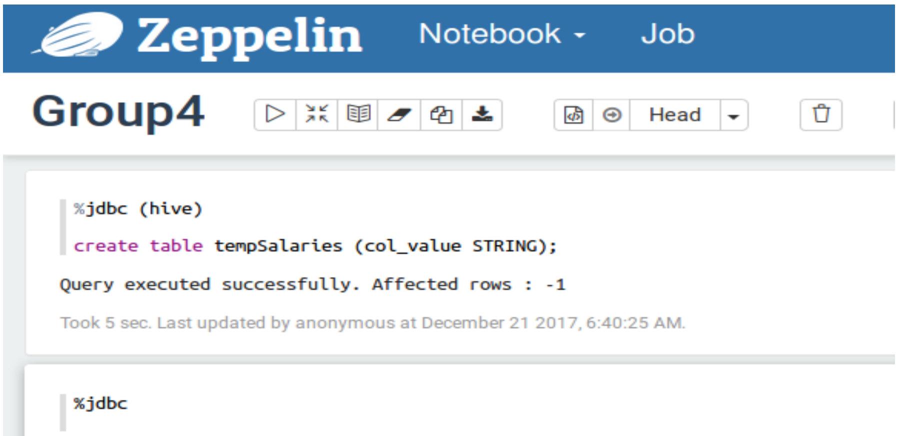


Figure 51: Create table tempSalaries

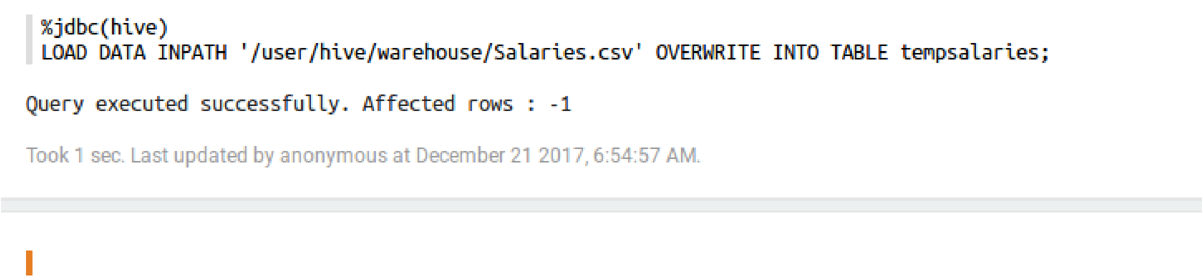


Figure 52: Load data tempSalaries in Zeppelin

To see the data:

```
%jdbc(hive)
select *from tempsalaries limit 5;
```

tempsalaries.col_value
yearID,teamID,lgID,playerID,salary
1985,BAL,AL,murraed02,1472819
1985,BAL,AL,lynnfr01,1090000
1985,BAL,AL,ripkeca01,800000
1985,BAL,AL,lacyle01,725000

Figure 53: View data tempSalaries in Zeppelin

Create Table for Salaries

```
%jdbc(hive)
CREATE TABLE salaries (yearID INT, teamID STRING, igID STRING, playerID STRING, salary INT);

Query executed successfully. Affected rows : -1
```

Took 1 sec. Last updated by anonymous at December 21 2017, 7:02:37 AM.

Figure 54: Create table salaries in Zeppelin

Extract data from tempSalaries and store them into Salaries table

```
%jdbc(hive)
insert overwrite table salaries SELECT

regexp_extract(col_value, '^(?:([^,]*),?){1}', 1) yearID,

regexp_extract(col_value, '^(?:([^,]*),?){2}', 1) teamID,

regexp_extract(col_value, '^(?:([^,]*),?){3}', 1) igID,

regexp_extract(col_value, '^(?:([^,]*),?){4}', 1) playerID,

regexp_extract(col_value, '^(?:([^,]*),?){5}', 1) salary

from tempsalaries;

Query executed successfully. Affected rows : -1
```

Took 25 sec. Last updated by anonymous at December 21 2017, 7:09:02 AM.

Figure 55: Extract data from tempSalaries in Zeppelin

Create tempBatting and load batting.csv

```
%jdbc(hive)
create table tempbatting (col_value STRING);
LOAD DATA INPATH '/user/hive/warehouse/Batting.csv' OVERWRITE INTO TABLE tempbatting;

Query executed successfully. Affected rows : -1

Query executed successfully. Affected rows : -1

Took 1 sec. Last updated by anonymous at December 21 2017, 7:20:02 AM.
```

Figure 56: Create table and load data into tembatting

Create the table Batting

create table batting (playerID String, yearID int, stint int, teamID String, IgID String, G int, G_batting int, AB int, R int, H int, n2B int, n3B int, HR int);

```
%jdbc(hive)
create table batting (playerID String, yearID int, stint int, teamID String, IgID String, G int, G_batting int, AB int, R int, H int, n2B int, n3B int, HR int);
insert overwrite table batting
SELECT
regexp_extract(col_value, '^(?:([^,]*),?){1}', 1) playerID,
regexp_extract(col_value, '^(?:([^,]*),?){2}', 1) yearID,
regexp_extract(col_value, '^(?:([^,]*),?){3}', 1) stint,
regexp_extract(col_value, '^(?:([^,]*),?){4}', 1) teamID,
regexp_extract(col_value, '^(?:([^,]*),?){5}', 1) IgID,
regexp_extract(col_value, '^(?:([^,]*),?){6}', 1) G,
regexp_extract(col_value, '^(?:([^,]*),?){7}', 1) G_batting,
regexp_extract(col_value, '^(?:([^,]*),?){8}', 1) AB,
regexp_extract(col_value, '^(?:([^,]*),?){9}', 1) R,
regexp_extract(col_value, '^(?:([^,]*),?){10}', 1) H,
regexp_extract(col_value, '^(?:([^,]*),?){11}', 1) n2B,
regexp_extract(col_value, '^(?:([^,]*),?){12}', 1) n3B,
regexp_extract(col_value, '^(?:([^,]*),?){13}', 1) HR
from tempbatting;
Query executed successfully. Affected rows : -1

Query executed successfully. Affected rows : -1
```

Figure 57: Extract data from tempbatting in Zeppelin

To see the batting data:

```
%jdbc(hive)
select *from batting limit 5;
```

FINISHED

batting.playerid	batting.yearid	batting.stint	batting.teamid	batting.igid	batting.g	batting.g_batting	batting.ab	batting.r	batting.h	batting.n2b	batting.n3
playerID	null	null	teamID	IgID	null	null	null	null	null	null	null
aardsda01	2004	1	SFN	NL	11	11	0	0	0	0	0
aardsda01	2006	1	CHN	NL	45	43	2	0	0	0	0
aardsda01	2007	1	CHA	AL	25	2	0	0	0	0	0
aardsda01	2008	1	BOS	AL	47	5	1	0	0	0	0

Figure 58: View data

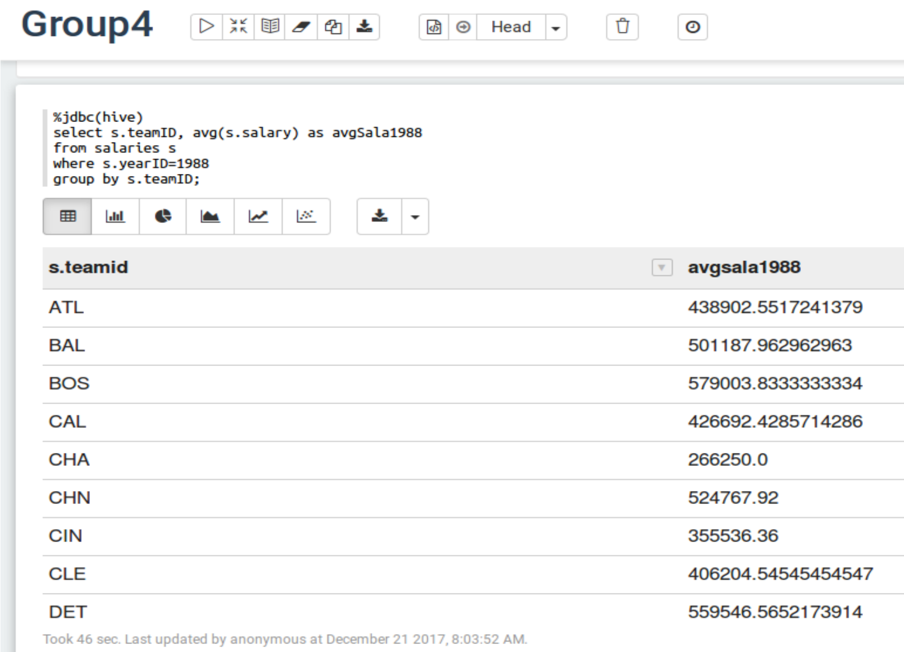
B. Print the names of all teams that every player had a salary higher than 500,000$ in the year of 2000.

```
%jdbc(hive)
select distinct s1.teamid

from salaries s1

where s1.yearid=2000

and not exists (

select *

from salaries s2

where s2.salary<=500000

and s2.yearid=2000

and s2.teamid=s1.teamid);
```

s1.teamid

Figure 59: Task 1 in Zeppelin

C. Print all the teams with their corresponding average salary in 1988.



Group4

```
%jdbc(hive)
select s.teamID, avg(s.salary) as avgSala1988
from salaries s
where s.yearID=1988
group by s.teamID;
```

s.teamid	avgsala1988
ATL	438902.5517241379
BAL	501187.962962963
BOS	579003.8333333334
CAL	426692.4285714286
CHA	266250.0
CHN	524767.92
CIN	355536.36
CLE	406204.54545454547
DET	559546.5652173914

Took 46 sec. Last updated by anonymous at December 21 2017, 8:03:52 AM.

Figure 60: Task 2 in Zeppelin

Visualization

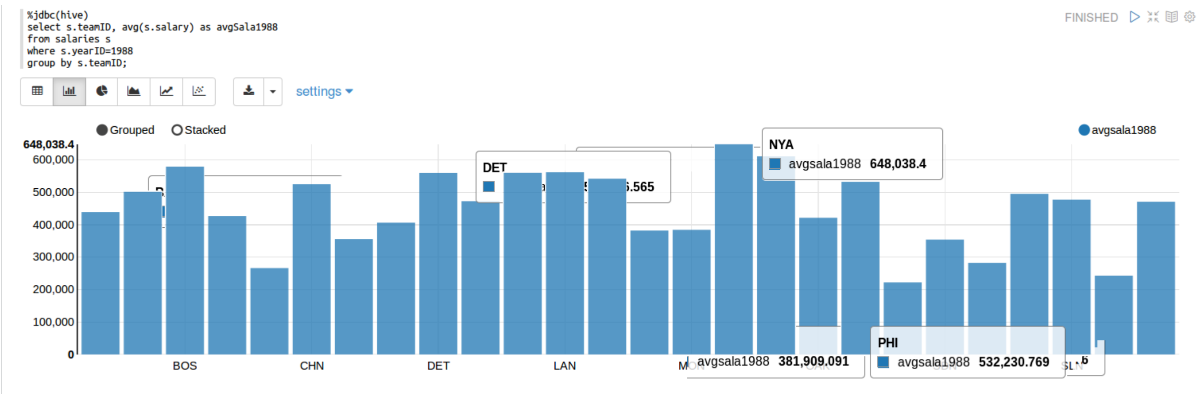


Figure 61: Task 2 view in Zeppelin

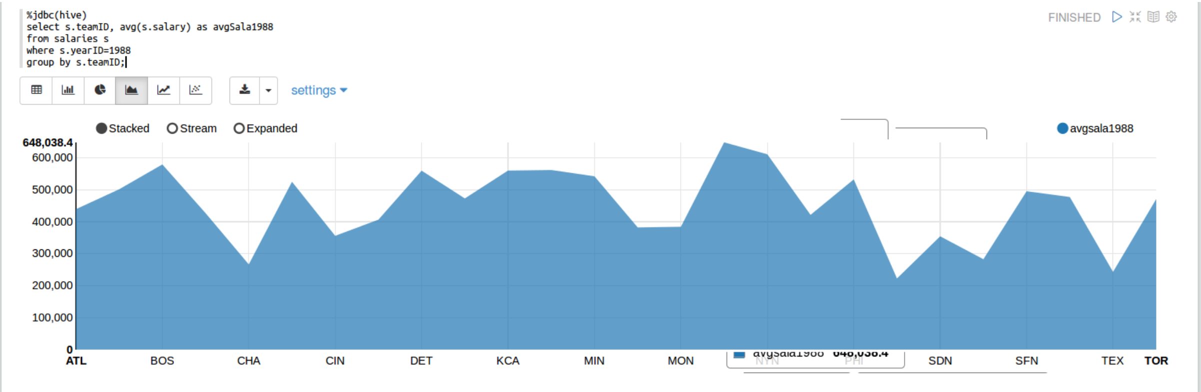


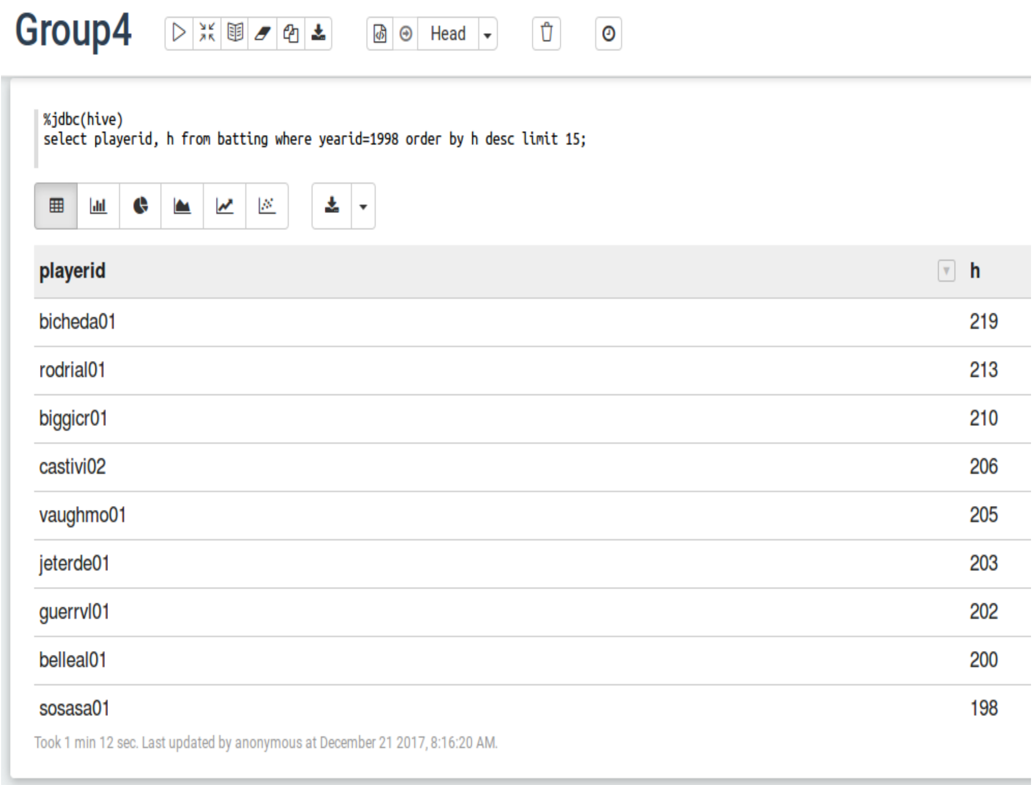
Figure 62: Task 2 view 2 in Zeppelin

D. Print the 15 players who made the most hit in 1998. The display must be the

5PlayerID and the amount of hits.

%jdbc(hive)

Select playerid, h from batting where yearid = 1998 order by h desc limit 15;



playerid	h
bicheda01	219
rodrial01	213
biggicr01	210
castivi02	206
vaughmo01	205
jeterde01	203
guerrvl01	202
belleal01	200
sosasa01	198

Took 1 min 12 sec. Last updated by anonymous at December 21 2017, 8:16:20 AM.

Figure 62: Task 2 result in Zeppelin

Visualization with Zeppelin

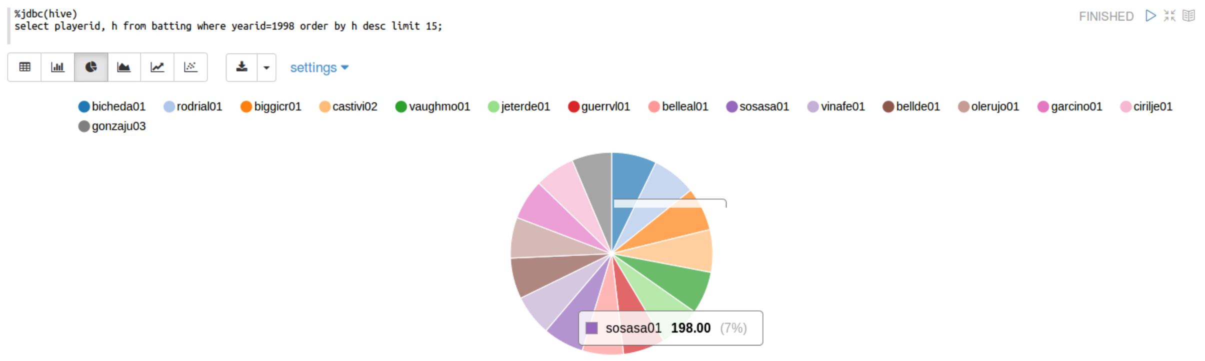


Figure 63: Task 3 view in Zeppelin

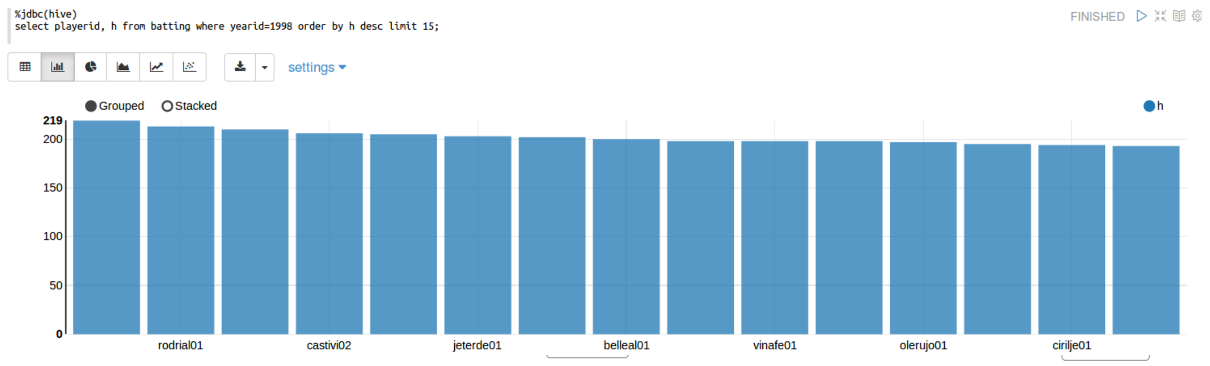


Figure 64: Task 2 view in Zeppelin

E. Print the player in the ML1 team with the most runs in 1960. The display must be the PlayerID and the number of runs.

%jdbc (hive)

Select playerid, r from batting where yearid = 1960 and teamid = "ML1" sort by r desc limit 1;

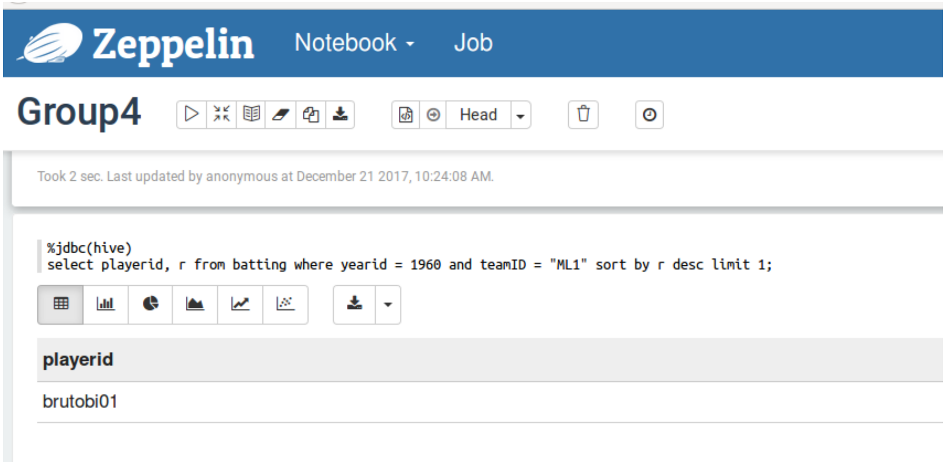


Figure 65: Task 4 result in Zeppelin

F. Print those players with a salary above 500,000$ in 2002 who have made more than 30 homeruns. Display the players, their amount of homeruns and salary.

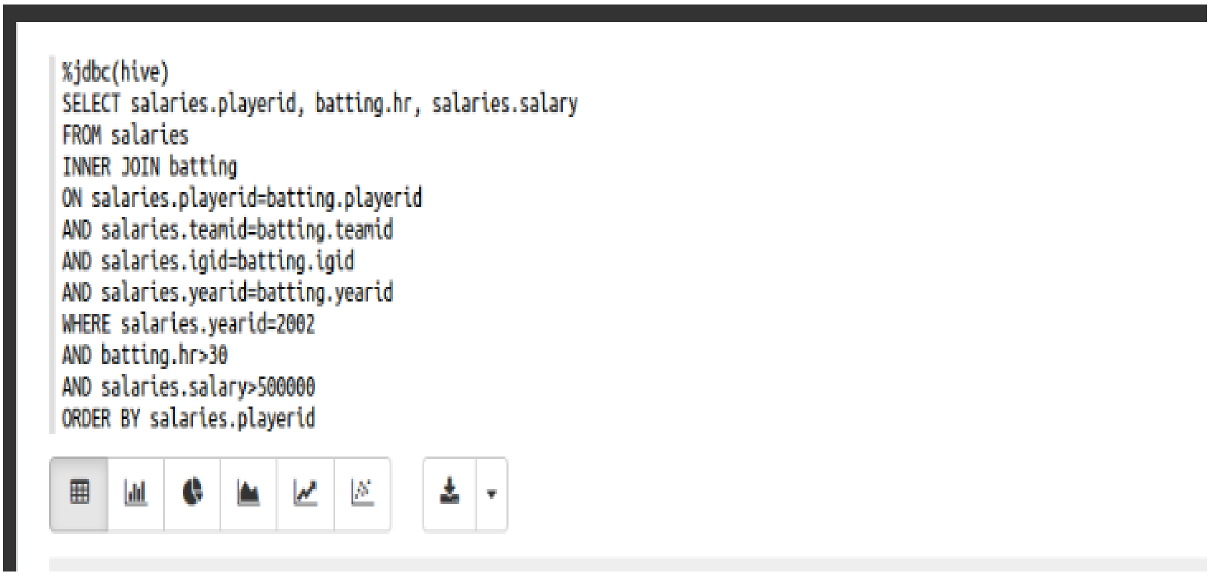


Figure 66: Task 5 in Zeppelin

Results:

```
bat.playerid,bat.hr,sal.salary
bagweje01,31,11000000
batisto01,31,4900000
bondsba01,46,15000000
burksel01,32,6666667
burrepa01,37,1905000
chaveer01,34,2125000
delgaca01,33,19400000
giambja01,41,10428571
gilesbr02,38,8063003
greensh01,42,13416667
guerrvl01,39,8000000
jonesan01,35,10000000
kentje01,37,6000000
ordonma01,38,6500000
palmera01,43,8712986
piazzmi01,33,10571429
pujolal01,34,600000
ramirma02,33,15462727
rodrial01,57,22000000
soriaal01,39,630000
sosasa01,49,15000000
tejadmi01,34,3625000
thomeji01,52,8000000
```

Figure 67: Task 5 result in Zeppelin

Visualization with Zeppelin

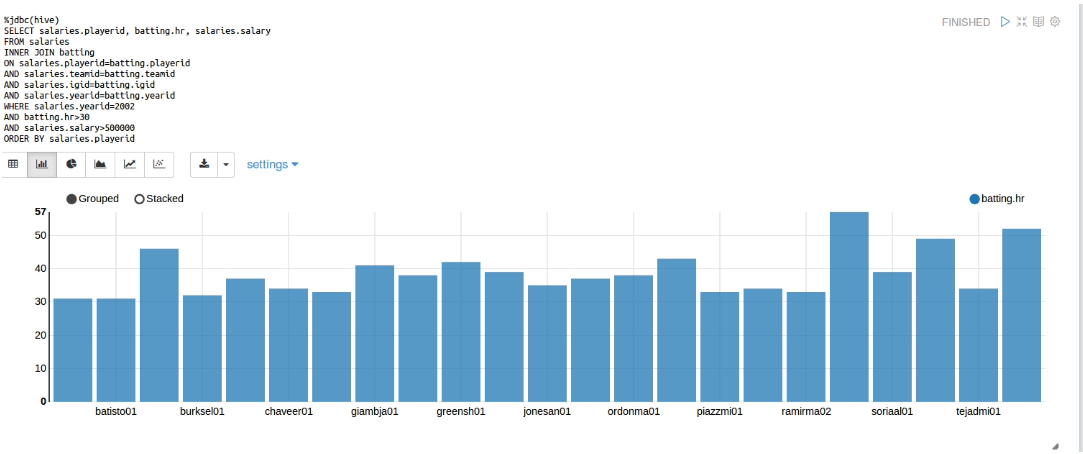


Figure 68: Task 5 view in Zeppelin

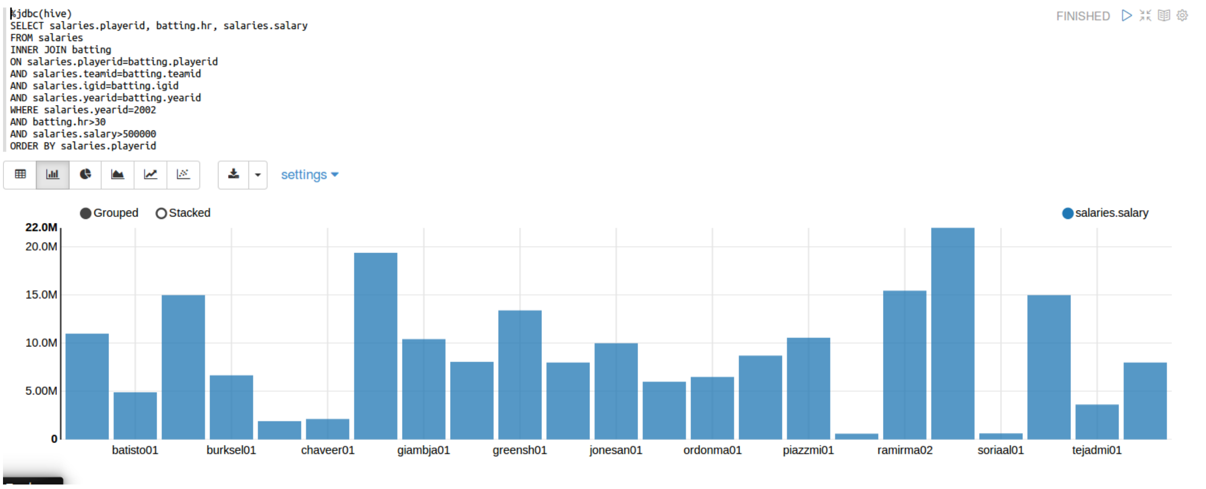


Figure 69: Task 5 view 2 in Zeppelin

Task 3: Advanced Analytics algorithms using SparkR

3.1 Perform Summary, Linear, Regression analysis, Logistic Regression Analysis and K-mean Analysis

We followed the steps recommended in “CN7022 Lab session 3 for 15/December2017” to do Tasks (3) SparkR. As: 1/ using byte columns and columns protocol(char), flag(char) and urgent(numeric) had been proposed; 2/ values of some columns are highly correlated (e.g. num_compromised and num_root: 0.9959602533); 3/ values in some columns are nearly no change (e.g. variance of num_out_cmds: 0); 4/ our present knowledge prevents us to convert columns protocol and flag to meaningful numeric for performing analysis, we finally decided to choose the following columns for further studies unless stated otherwise:

“src_bytes”,”dst_bytes”,”urgent”,”same_srv_rate”,”diff_srv_rate”,”count”,”srv_count”,”dst_host_srv_count”,”dst_host_same_srv_rate”,”attacks” and “attacks_code”

(Note: values in column attacks are characters of “normal.” Or other names, a new numeric column attacks_code was created with items corresponding to “normal.” Converted to 0 and others converted to 1.

Procedures:

#import sparklyr and dplyr

$library(sparklyr)

$library(dplyr)

#activate sparklyr

$sc <- spark_connect(master = “local”)

```
#import data (from full data line 2 to line 1048576)

#filename and its path are vary

$kdd0 <- read.csv("kddcup.data.version.used.csv", header = T)

#adding attacks_code (0 or 1)

$kdd0$attacks_code<-ifelse(kdd0$attacks=="normal.", 0, 1)

#selecting those columns as mentioned above for further analysis

$kdd<-
kdd0[,c("src_bytes","dst_bytes","urgent","same_srv_rate","diff_srv_rate","count","srv_count","dst_host_srv_count",
"dst_host_same_srv_rate","attacks","attacks_code")]

#creating sparklyr tibbles

$kdd_tbl <- sdf_copy_to(sc = sc, x = kdd, overview = T)

#partitioning kdd_tbl into a training set(80%) and a test set (20%)/ seed:423

$partition <- kdd_tbl %>%

+partition <- kdd_tbl %>%

$train_kdd_tbl <- partition$train

$test_kdd_tbl <- partition$test

#choose summary_same_srv_rate to perform the summarize() function

$kdd_summary_same_srv_rate<-kdd_tbl %>%

+ group_by(attacks, count) %>%

+ summarize(countNo=n(), mean_same_srv_rate=mean(same_srv_rate), stdev_same_srv_rate=sd(same_srv_rate))%>%

+ collect

$print(kdd_summary_same_srv_rate)

> print(kdd_summary_same_srv_rate)

# A tibble: 2,051 x 5

# Groups:   attacks [20]

   attacks count countNo mean_same_srv_rate stdev_same_srv_rate



  <chr> <int>   <dbl>            <dbl>             <dbl>

 1 normal.    2   51669         0.9727786         0.113445682

 2 normal.    5   28726         0.9978904         0.035436927

 3 normal.   17    9462         0.9991482         0.025865699

 4 normal.   21    5956         0.9992629         0.023865226

 5 normal.   27    3467         0.9996106         0.015381045

 6 normal.   30    2416         0.9999379         0.001363634

 7 normal.   32    2095         0.9990167         0.029537783

 8 normal.   35    1539         0.9987654         0.031580128

 9 normal.   38     992         0.9982258         0.036189286

10 normal.   39     894         0.9980872         0.036935057
```

... with 2,041 more rows

#plot the summary kdd_summary_same_srv_rate

#note: ggplot2 must first be installed and imported to the library

$library(ggplot2)

$ggplot(kdd_summary_same_srv_rate, aes(attacks, count, color=attacks))+

+ geom_line(size=1.2) +

+ geom_errorbar(aes(ymin=mean_same_srv_rate – stdev_same_srv_rate, ymax =

+ mean_same_srv_rate + stdev_same_srv_rate), width = 0.05)+

+ geom_text(aes(label=countNo), vjust=-0.2, color="black")+

+ theme(legend.position = "top")

##the graph is attached

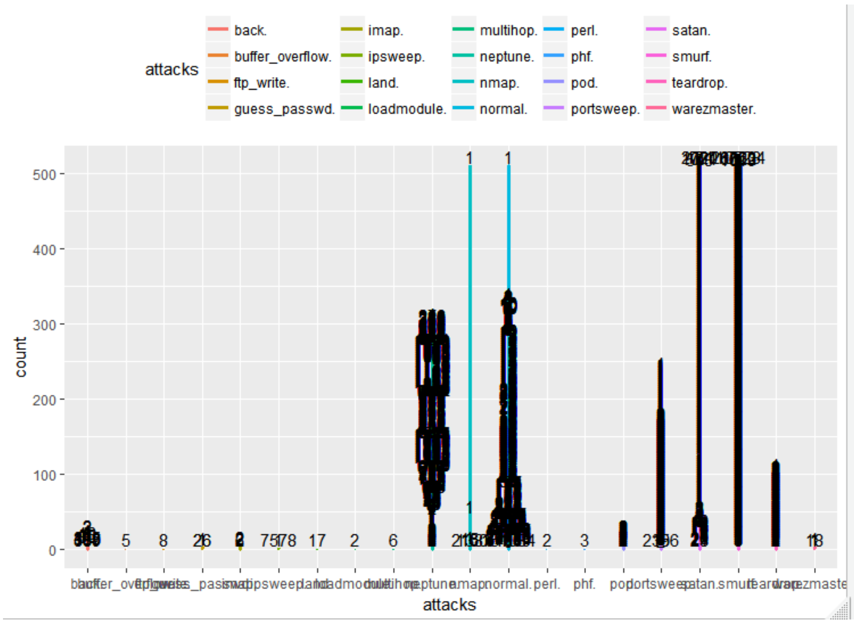


Figure 70: Plot of Count Vs Attack

#perform logistic regression analysis

$logistic_train_kdd <- ml_logistic_regression(x=train_kdd_tbl, response = "attacks_code", features = c("src_bytes", "dst_bytes", "urgent", "count", "srv_count", "same_srv_rate", "diff_srv_rate", "dst_host_srv_count", "dst_host_same_srv_rate"))

$print(logistic_train_kdd)

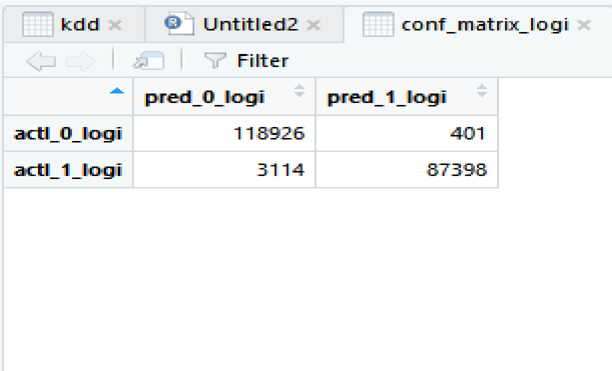
Call: attacks_code ~ src_bytes + dst_bytes + urgent + count + srv_count + same_srv_rate + diff_srv_rate + dst_host_srv_count + dst_host_same_srv_rate

Coefficients:

(Intercept)	src_bytes
6.917022e+00	2.736398e-07

```
       dst_bytes            urgent
     2.359772e-07       -3.775281e-02
           count           srv_count
     5.690396e-03        1.517534e-02
     same_srv_rate        diff_srv_rate
     -8.460347e+00       -6.005035e+00
 dst_host_srv_count dst_host_same_srv_rate
     -9.650259e-03       -7.472369e-01

#use test set to perform testing
$prediction_logi = collect(sdf_predict(logistic_train_kdd, test_kdd_tbl))

#for logistic regression analysis, sparklyr only provide predict results at
#either 0 or 1, therefore, cut-off(threshold) is irrelevant.
#Without cut-off, at the moment we are not capable of performing
#ROC(AUC) analysis. Therefore, in logistic regression analysis,
#we only perform confusion matrix analysis

#create a table of two columns(actual attack_code and prediction)
#for performing confusion matrix analysis.
#This table may not be required for experienced R programmers

$form_conf_matx_logi<-test_kdd_tbl%>%
+ select(actl_attks_code=attacks_code)%>%
+ collect()%>%
+ mutate(pred_attks_code=prediction_logi$prediction)%>%
+ collect()

$form_conf_matx_logi$actl_attks_code<-ifelse(form_conf_matx_logi$actl_attks_code==0,"0", "1")

$form_conf_matx_logi$pred_attks_code<-ifelse(form_conf_matx_logi$pred_attks_code==0,"0", "1")

$form_conf_matx_logi$actl_pred<-paste(form_conf_matx_logi$actl_attks_code,
form_conf_matx_logi$pred_attks_code, sep="_")

$TP_logi<-subset(form_conf_matx_logi, subset =actl_pred=="0_0")
$TP_logi_no<-count(TP_logi)
$TP_logi_num<-TP_logi_no$n    #118926
$TN_logi<-subset(form_conf_matx_logi, subset =actl_pred=="1_1")
```

```
$TN_logi_no<-count(TN_logi)

$TN_logi_num<-TN_logi_no$n    #87398

$FP_logi<-subset(form_conf_matx_logi, subset =actl_pred==”1_0”)

$FP_logi_no<-count(FP_logi)

$FP_logi_num<-FP_logi_no$n    #3114

$FN_logi<-subset(form_conf_matx_logi, subset =actl_pred==”0_1”)

$FN_logi_no<-count(FN_logi)

$FN_logi_num<-FN_logi_no$n    #401


#form confusion matrix dataframe

$pred_0_logi<-c(TP_logi_num, FP_logi_num)

$pred_1_logi<-c(FN_logi_num, TN_logi_num)

$conf_matrix_logi<-data.frame(“pred_0_logi”=pred_0_logi,”pred_1_logi”=pred_1_logi)

#add row names

$row.names(conf_matrix_logi) <- c(“actl_0_logi”,”actl_1_logi”)


$View(conf_matrix_logi)

##the 2*2 dataframe is attached.
```



	pred_0_logi	pred_1_logi
actl_0_logi	118926	401
actl_1_logi	3114	87398

Figure 71: The 2*2 dataframe

There is an interesting finding: even with the same dataset, the same proportion of training set and test set, and the same seed number (here we use 423) for dividing the data, the datasets obtained by using different computer would differ slightly. For example:

Groupmate 1's pc: test set row: 209839, TP: 118926, TN: 87398, FP: 3114, FN: 401

Groupmate 2's pc: test set rowL 210299, TP: 119155, TN: 87579, FP: 3172, FN: 393

The different in number of rows: (210299-209839)/209839 = 0.000001044%

Perhaps it is because of rounding errors

#Calculation of precision, specificity, sensitivity(recall) and accuracy

#Precision(PPV) = TP/ (TP+FP) ………………… #0.9740695

$precision_logi<-TP_logi_num/(TP_logi_num+FP_logi_num)

$precision_logi

#Specificity(TNR) = TN/ (TN+FP)………………. #0.9650472

$specificity_logi<-TN_logi_num/(TN_logi_num+FP_logi_num)

$specificity_logi

#Sensitivity(Recall/ TPR) = TP/(TP+FN) #0.9967126

$sensitivity_logi<-TP_logi_num/(TP_logi_num+FN_logi_num)

$sensitivity_logi

#Accuracy(ACC) = (TP+TN)/ (TP+FP+FN+TN) #0.9830479

$accuracy_logi<-(TP_logi_num+TN_logi_num)/(TP_logi_num+FP_logi_num+FN_logi_num+TN_logi_num)

$accuracy_logi

#the confusion matrix and the values derived from it indicate that

#the model is reliable. However, further analyses should be performed to

#find out why the results are so good. As mentioned above, ROC(AUC) analysis

#would not be conducted in respect of logistic regression since predictions

#given are either 0 or 1. It makes cut-off(threshold) irrelevant.

#perform linear regression analysis

$linear_train_kdd <- ml_linear_regression(x=train_kdd_tbl, response = “attacks_code”, features = c(“src_bytes”, “dst_bytes”, “urgent”, “count”, “srv_count”, “same_srv_rate”, “diff_srv_rate”, “dst_host_srv_count”, “dst_host_same_srv_rate”))

$print(linear_train_kdd)

Call: ml_linear_regression(x = kdd_tbl, response = “attacks_code”, features = c(“src_bytes”, “dst_bytes”, “urgent”, “count”, “srv_count”, “same_srv_rate”, “diff_srv_rate”, “dst_host_srv_count”, “dst_host_same_srv_rate”))

Coefficients:

(Intercept)	**src_bytes**
1.025382e+00	**2.737760e-10**
dst_bytes	**urgent**
4.038690e-08	**3.828738e-03**
count	**srv_count**
1.967680e-04	**1.785656e-03**
same_srv_rate	**diff_srv_rate**
-8.816017e-01	**-2.942815e-01**
dst_host_srv_count	**dst_host_same_srv_rate**
-3.356343e-04	**-7.432852e-02**

#use test set to perform testing

$prediction_linr = collect(sdf_predict(linear_train_kdd, test_kdd_tbl))

```
#create a table of two columns(actual attack_code and prediction)

#for performing confusion matrix analysis.

#This table may not be required for experienced R programmers.

$form_conf_matx_linr<-test_kdd_tbl%>%

+ select(actl_attks_code=attacks_code)%>%

+ collect()%>%

+ mutate(pred_attaks_code=prediction_linr$prediction)%>%

+ collect()
```

$head(form_conf_matx_linr)

> head(form_conf_matx_linr)

```
# A tibble: 6 x 2
  actl_attks_code pred_attaks_code
            <dbl>            <dbl>
1               1        0.8758290
2               1        0.8701314
3               1        0.8764280
4               1        0.8723046
5               1        0.8770270
6               1        0.8729036
```

```
#unlike in logistic regression analysis where predictions provided by

#sparklyr are either one or zero, in linear regression analysis

#predictions are not in whole numbers. Therefore, ROC(AUC) analyses

#will be conducted hereunder. However, confusion matrix analysis will

#not be performed in respect of linear regression analysis


#install pROC package for ROC(AUC) analysis

$install.packages("pROC")

$library(pROC)

$ROC_linr <- roc(form_conf_matx_linr$actl_attks_code, form_conf_matx_linr$pred_attaks_code)    #this command take some time

$plot(ROC_linr)

##picture of the graph is attached
```

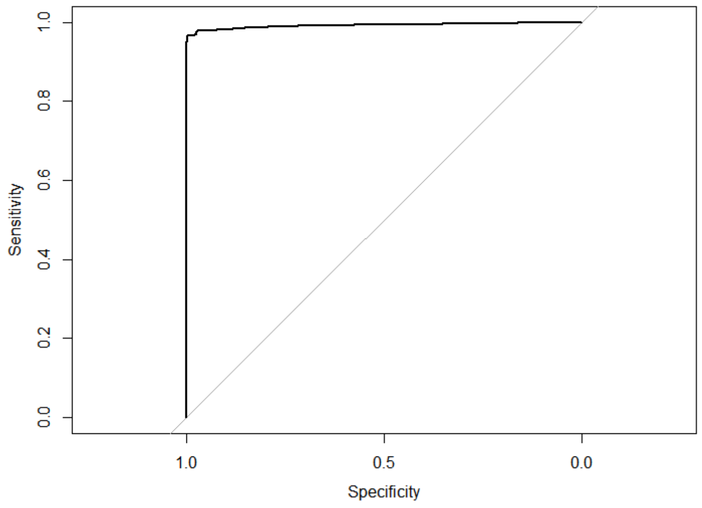


Figure 72: ROC graph

$AUC_linr<-auc(form_conf_matx_linr$actl_attks_code,   form_conf_matx_linr$pred_attaks_code)   #this command take some time

$AUC_linr   #0.991

#AUC of linear regression analysis is very close to 1.

#Further studies should be conducted to find out why the results are so perfect.

#perform kmeans clustering analysis

#for experiencing visualisation of kmeans clustering, a 2D kmeans analysis

#is conducted below (by using these two columns: count and same_srv_rate

$kmeans_2D_train_kdd<-ml_kmeans(x=train_kdd_tbl, centers = 3, features = c("count","same_srv_rate"))

$prediction_kmeans_2D = collect(sdf_predict(kmeans_2D_train_kdd, test_kdd_tbl))

$kmeans_2D_plot<-prediction_kmeans_2D %>%

+ ggplot(aes(count, same_srv_rate)) +

+ geom_point(aes(count, same_srv_rate, col = factor(prediction + 1)),

+ size = 2, alpha = 0.5) +

+ geom_point(data = kmeans_2D_train_kdd$centers, aes(count, same_srv_rate),

+ col = scales::muted(c("red", "green", "blue")),

+ pch = 'x', size = 12) ++ scale_color_discrete(name = "Predicted Cluster",

+ labels = paste("Cluster", 0:2)) +

+ labs(

+ x = "count",

+ y = "same_srv_rate",

+ title = "2D K-Means Clustering",

+ subtitle = "use Spark.ML to predict cluster membership with the KDD dataset\n (2-dimensional k-means for observing visualization of k-means clustering)")

#view the plot

$kmeans_2D_plot

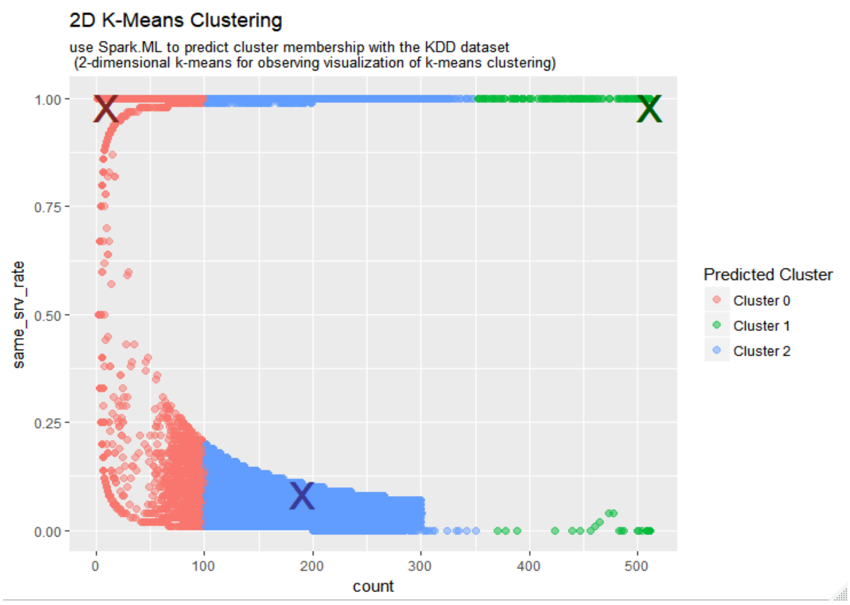


Figure 73: kmeans_2D_plot

#multi-dimentional kmeans

$kmeans_train_kdd <- ml_kmeans(x=train_kdd_tbl, centers = 3, features = c(“src_bytes”,”dst_bytes”,”urgent”,”same_srv_rate”,”diff_srv_rate”,”count”,”srv_count”,”dst_host_srv_count”,”dst_host_same_srv_rate”,”attacks_code”))

$print(kmeans_train_kdd)

K-means clustering with 3 clusters

Cluster centers:

src_bytes dst_bytes urgent same_srv_rate diff_srv_rate count srv_count

1 9.214534e+02 2365.767 2.147277e-05 0.8092967 0.02125147 154.6181 120.2846

2 6.933756e+08 0.000 0.000000e+00 0.0500000 0.39000000 57.0000 3.0000

3 1.195406e+07 1152524.000 0.000000e+00 1.0000000 0.00000000 1.2500 1.2500

dst_host_srv_count dst_host_same_srv_rate attacks_code

1 186.4497 0.7601483 0.4318723

2 3.0000 0.0100000 1.0000000

3 2.0000 0.1025000 0.0000000

Within Set Sum of Squared Errors = 1.473057e+15

#perform kmeans analysis by using the test set data

$kmeans_test_kdd <- ml_kmeans(x=test_kdd_tbl, centers = 3, features = c(“src_bytes”,”dst_bytes”,”urgent”,”same_srv_rate”,”diff_srv_rate”,”count”,”srv_count”,”dst_host_srv_count”,”dst_host_same_srv_rate”,”attacks_code”))

$print(kmeans_test_kdd)

```
K-means clustering with 3 clusters

Cluster centers:

    src_bytes   dst_bytes      urgent same_srv_rate

1 9.141093e+02    2063.825 9.511399e-06     0.8101834

2 5.270000e+02 1984880.458 0.000000e+00     1.0000000

3 1.458409e+07       0.000 0.000000e+00     1.0000000

  diff_srv_rate      count  srv_count

      3      0.02085008 154.732411 120.513887

2    0.00000000   1.416667   1.416667

3    0.00000000   2.000000   2.000000

  dst_host_srv_count dst_host_same_srv_rate

1          186.60177              0.7607367

2           56.20833              0.3987500

3           17.00000              0.0700000

  attacks_code

1    0.43157499

2   0.08333333

3   0.00000000


Within Set Sum of Squared Errors = 1.648048e+14

#Predict test set data by using kmeans analysis result conducted with training set

$prediction_kmeans_train_test = collect(sdf_predict(kmeans_train_kdd, test_kdd_tbl))


#predict test set data by using kmeans analysis result conducted

#with test set itself

$prediction_kmeans_test_test = collect(sdf_predict(kmeans_test_kdd, test_kdd_tbl))

#create a table of two columns(pred_kmeans_testtest and pred_kmeans_traintest)

#for performing confusion matrix analysis

#We are not sure whether this comparison is meaningful, correct or useful

$form_conf_matx_kmeans<-prediction_kmeans_test_test%>%

+ select(pred_kmeans_testtest=prediction)%>%

+ collect()%>%

+ mutate(pred_kmeans_traintest=prediction_kmeans_train_test$prediction)%>%

+ collect()

$form_conf_matx_kmeans$pred_kmeans_testtest<-ifelse(form_conf_matx_kmeans$pred_kmeans_testtest==0,"0", "1")

$form_conf_matx_kmeans$pred_kmeans_traintest<-ifelse(form_conf_matx_kmeans$pred_kmeans_traintest==0,"0", "1")

$form_conf_matx_kmeans$actl_pred<-paste(form_conf_matx_kmeans$pred_kmeans_testtest,
form_conf_matx_kmeans$pred_kmeans_traintest, sep="_")

$TP_kmeans<-subset(form_conf_matx_kmeans, subset =actl_pred=="0_0")
```

```
$TP_kmeans_no<-count(TP_kmeans)
$TP_kmeans_num<-TP_kmeans_no$n
$TP_kmeans_num    #210274
$TN_kmeans<-subset(form_conf_matx_kmeans, subset =actl_pred==”1_1”)
$TN_kmeans_no<-count(TN_kmeans)
$TN_kmeans_num<-TN_kmeans_no$n
$TN_kmeans_num    #1
$FP_kmeans<-subset(form_conf_matx_kmeans, subset =actl_pred==”1_0”)
$FP_kmeans_no<-count(FP_kmeans)
$FP_kmeans_num<-FP_kmeans_no$n
$FP_kmeans_num    #24
$FN_kmeans<-subset(form_conf_matx_kmeans, subset =actl_pred==”0_1”)
$FN_kmeans_no<-count(FN_kmeans)
$FN_kmeans_num<-FN_kmeans_no$n
$FN_kmeans_num    #0
#predict() was used to    llocate test set data to kmean clusters found
#with test set itself and found with training set and found
#TP=210274, TN=1, FP=24, FN=0
#predict() was used to allocate test set data to kmean clusters found
#with test set itself and found with training set and found
#TP=210274, TN=1, FP=24, FN=0
#further studies should be conducted to find out whether this comparison is
#meaningful and useful and if yes, implications of the results
```

4.0 Individual Assessment

4.1 John Olorunsuyi

At the end of the module, I have acquired the good understanding of Hadoop and its eco-systems like MapReduce, Pig, Sqoop, Flume, and Hive. Also learnt how to install and configure Apache Ambari on multi-nodes, for providing, managing and monitoring Hadoop services across clusters. I was able use Apache Hive on Ambari web view to analysed dataset. Apache Zeppelin was used to visualise the results various analysis done. I now good understanding of data analysis using SparkR, which is a predictive tool and for machine learning.

- Alternative technology Apache Ambari is the Cloudera Managers
- Apache Pig is alternative to Hive and
- R is an alternative Zeppelin
- MATLAB can be used as alternative to SparkR, but MATLAB is not open source and would require license while SparkR is an open source software

4.2 Chung Hoi Chiu

1/ What did you learn from the coursework
The coursework introduced us four tools: Ambari, Hive, Zeppelin and sparklyr. The coursework is especially valuable in guiding us to install Ambari. I have been informed that install and configure software form an initial hurdle for learning big data analytics.

2/ Alternative technologies

Ambari
Ambari is an open source management tool alternative to Cloudera Manager. According to what people said in Quora, users of Ambari are especially impressed by its ‘complete roll-back capability’ in the configuration stage and its customisable UI dashboard. Its open source nature also make it free from vendor lock-in.

Hive
The alternative to Hive is Pig. Both these two tools aim to replace the challenging work of writing MapReduce code. However, while Hive adopts the usual SQL language, Pig has its own language called Pig Latin.

Zeppelin as a visualisation tool
Zeppelin is not just a data visualisation tool; it is a multi-purpose notebook which can perform data ingestion, data discovery, data analytics, data visualisation and collaboration. For visualising plots with R code in Zeppelin, one needs to install in Zeppelin R packages for visualisation of which ggplot2 (using it is required in the coursework) is an important one.

Sparklyr
The alternative to the R package sparklyr is R itself. However, in R all data must be loaded to the RAM of a single computer for further analysis. By using sparklyr, which grants access to Spark from R, one is able to obtain both goods: the language R for writing data analysis code quickly and readably, and Spark (a cluster computing platform) for handling huge datasets across multiple computers.

3/. New thinking evoked
New open source software in general lack easy-to-understand error messages. This phenomenon is especially challenging for novices. For example, as I could not understand the implication of the error message, I spent more than five hours trying to determine why I could not install sparklyr: sparklyr cannot recognise the apostrophe of the username of my computer “Eric’s Laptop”. Therefore, joining user communities and always remaining claim and patient should be essentials for learning big data analytics.

4.3 Teamwork minutes

Date of meeting	Minutes of meeting	Task allocation
November 18, 2017	Big stack installation planning. How many nodes would be installed	John and Eric
November 25, 2017	Hive Queries	Eric
December 2nd, 2017	Zeppelin installation	Eric
December 9th, 2017	Zeppelin Queries and visualisation	John
December 12tht, 2017	SparkR	Eric
December 18th, 2017	Preparation for presentation	John and Eric
December 23rd, 2017	Report layout	John

4.4 References

- O’Reilly Safari. (2018). *Hadoop Essentials*. [online] Available at: https://www.safaribooksonline.com/library/view/hadoop-essentials/9781784396688/ch02s05.html [Accessed 1 Jan. 2018].
- Techopedia Inc. (2018). *Home - Techopedia Inc.* [online] Available at: https://www.techopedia.com/definition/28659/big-data-analytics [Accessed 17 Dec. 2017].