



NITTE MEENAKSHI INSTITUTE OF TECHNOLOGY

An Autonomous Institution Approved by UGC/AICTE/Govt. of Karnataka

Accredited by NBA (Tier-I) and NAAC 'A+' Grade

Affiliated to Visvesvaraya Technological University, Belagavi

Post Box No. 6429, Yelahanka, Bengaluru-560064, Karnataka, INDIA

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

PRACTIAL RECORD BOOK

21ADL56: Big Data Analytics Laboratory

NAME :

USN :

YEAR/SEM : III/V

BRANCH : Artificial Intelligence and Data Science

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE



CERTIFICATE

This is to certify that Mr./Mrs. _____ studying in 5th semester Artificial Intelligence and Data Science Branch has satisfactorily completed the course of experiments in Big Data Analytics Laboratory for the academic session 2023-2024 as prescribed by the Visvesvaraya Technological University.

Register Number of the Candidate:

Signature of the Staff Incharge

Date

Signature of HOD

Date:

Sl. No	<u>Program / Exercise</u>
1.	Installation of Hadoop Framework, its components and study of the HADOOP ecosystem.
2.	Working with Hadoop Components -Hadoop Daemons and HDFS Commands
3.	Write a program to implement a word count program using MapReduce
4.	Write a program to analyse stock dataset using MapReduce
5.	Write a program to analyse temperature dataset using MapReduce
6.	Working with hive commands to analyse various partitioning, bucketing
7.	Working with hbase commands to analyse sample dataset
8.	Installation and configuration of spark, word count analysis using scala
9.	Implementing on Saving RDD - Lazy Operation - Spark Jobs-spark ML library.
10	Implement K-means Clustering algorithm using Map Reduce

INDEX SHEET

SL.NO	Programs	Page no	Date of Execution	Staff Signature
1.	Installation of Hadoop Framework, its components and study of the HADOOP ecosystem.			
2	Working with Hadoop Components -Hadoop Daemons and HDFS Commands			
3.	Write a program to implement a word count program using MapReduce			
4.	Write a program to analyse stock dataset using MapReduce			
5.	Write a program to analyse temperature dataset using MapReduce			
6.	Working with hive commands to analyse various partitioning, bucketing			
7.	Working with hbase commands to analyse sample dataset			
8.	Installation and configuration of spark, word count analysis using scala			
9.	Implementing on Saving RDD - Lazy Operation - Spark Jobs-spark ML library.			
10.	Implement K-means Clustering algorithm using Map Reduce			

Program 1	Installation of Hadoop Framework, its component and study of HADOOP ecosystem
Date-	

Prerequisites

- Access to a terminal window/command line
- Sudo or root privileges on local /remote machines

Install OpenJDK on Ubuntu

The Hadoop framework is written in Java, and its services require a compatible Java Runtime Environment (JRE) and Java Development Kit (JDK). Use the following command to update your system before initiating a new installation:

sudo apt update

Type the following command in your terminal to install OpenJDK 8:

sudo apt install openjdk-8-jdk -y

Once the installation process is complete, verify the current Java version:

java -version;

javac -version

Install OpenSSH on Ubuntu

Install the OpenSSH server and client using the following command:

sudo apt install openssh-server openssh-client -y

Create Hadoop User

sudo adduser hadoop

The username, in this example, is hadoop. You are free to use any username and password you see fit. Switch to the newly created user and enter the corresponding password

su – Hadoop

Download and Install Hadoop on Ubuntu

Visit the official Apache Hadoop project page, and select the version of Hadoop you want to implement.

Once the download is complete, extract the files to initiate the Hadoop installation:

```
tar xzf hadoop-3.2.1.tar.gz
```

The Hadoop binary files are now located within the *hadoop-3.2.1* directory.

Configure Hadoop Environment Variables (bashrc)

Edit the *.bashrc* shell configuration file using a text editor of your choice (we will be using nano)

```
sudo nano .bashrc
```

Define the Hadoop environment variables by adding the following content to the end of the file:

```
#Hadoop Related Options
```

```
export HADOOP_HOME=/home/hadoop/hadoop-3.2.1
```

```
export HADOOP_INSTALL=$HADOOP_HOME
```

```
export HADOOP_MAPRED_HOME=$HADOOP_HOME
```

```
export HADOOP_COMMON_HOME=$HADOOP_HOME
```

```
export HADOOP_HDFS_HOME=$HADOOP_HOME
```

```
export YARN_HOME=$HADOOP_HOME
```

```
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
```

```
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
```

```
export HADOOP_OPTS"-Djava.library.path=$HADOOP_HOME/lib/nativ"
```

Once you add the variables, save and exit the *.bashrc* file.

```
source ~/.bashrc
```

Edit *hadoop-env.sh* File

The *hadoop-env.sh* file serves as a master file to configure YARN, HDFS, MapReduce, and Hadoop-related project settings.

When setting up a **single node Hadoop cluster**, you need to define which Java implementation is to be utilized. Use the previously created *\$HADOOP_HOME* variable to access the *hadoop-env.sh* file:

```
sudo nano $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

Uncomment the `$JAVA_HOME` variable (i.e., remove the `#` sign) and add the full path to the OpenJDK installation on your system. If you have installed the same version as presented in the first part of this tutorial, add the following line:

```
export JAVA_HOME=/usr/lib/jvm/java-8-openjdk-amd64
```

Save and exit from env file

Edit `core-site.xml` File

The *core-site.xml* file defines HDFS and Hadoop core properties.

```
sudo nano $HADOOP_HOME/etc/hadoop/core-site.xml
```

Add the following configuration to override the default values for the temporary directory and add your HDFS URL to replace the default local file system setting:

```
<configuration>

<property>

  <name>hadoop.tmp.dir</name>

  <value>/home/hadoop/tmpdata</value>

</property>

<property>

  <name>fs.default.name</name>

  <value>hdfs://127.0.0.1:9000</value>

</property>

</configuration>
```

Save and exit the core file

Edit `hdfs-site.xml` File

The properties in the *hdfs-site.xml* file govern the location for storing node metadata, fsimage file, and edit log file. Configure the file by defining the **NameNode** and **DataNode storage directories**.

```
sudo nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

Add the following configuration to the file and, if needed, adjust the NameNode and DataNode directories to your custom locations:

```
<configuration>

<property>

  <name>dfs.data.dir</name>

  <value>/home/hadoop/dfsdata/namenode</value>

</property>

<property>

  <name>dfs.data.dir</name>

  <value>/home/hadoop/dfsdata/datanode</value>

</property>

<property>

  <name>dfs.replication</name>

  <value>1</value>

</property>

</configuration>
```

Save and exit the hdfs file

Edit mapred-site.xml File

Use the following command to access the *mapred-site.xml* file and **define MapReduce values**:

sudo nano \$HADOOP_HOME/etc/hadoop/mapred-site.xml

Add the following configuration to change the default MapReduce framework name value to yarn:

```
<configuration>

<property>

  <name>mapreduce.framework.name</name>
```


<value>yarn</value>

</property>

</configuration>

RESULT: Installation of Hadoop Framework, its component and the study of HADOOP ecosystem was completed successfully

Program 2	Working with Hadoop components -Hadoop Daemons and HDFS commands
Date-	

Hadoop Daemons

Hadoop has 5 daemons.They are NameNode, DataNode, Secondary NameNode, JobTracker and TaskTracker.

- **NameNode**
It stores the Meta Data about the data that are stored in DataNodes.
- **DataNode**
It stores the actual Data.
- **Secondary NameNode**
It is the backup of namenode and it just contains entire metadata of data nodes like data node address, properties and block report of each data node.
- **JobTracker**
Job Tracker is a master which creates and runs the job. JobTracker that runs on name node, allocates the job to TaskTrackers.
- **TaskTracker**
TaskTracker is a slave and runs on data node. TaskTracker runs the tasks and reports the status of task to JobTracker.

HDFS commands

1. To get the list of all the files in the HDFS root directory

- Command: Usage: hdfs dfs [generic options] -ls [-c] [-h] [-q] [-R] [-t] [-S] [-u] [<path>...]
- Note: Here, choose the path from the root, just like the general Linux file system. -h in Green Mark shows that it is in human-readable sizes, as recommended. -R in Blue Mark shows that it is different from numerous one to practice into subdirectories.

2. Help

- Command: fs -help
- Note: It prints the long output which prints all the commands

3. Concatenate all the files into a catalogue within a single file

- Command: `hdfs dfs [generic options] -getmerge [-nl] <src> <localdst>`
- Note: This will generate a new file on the local system directory which carries all files from a root directory and concatenates all together. -nl option, which is marked in Red, combines newlines among the files. With the help of this command, you can combine a collection of small records within a selection for a different operation.

4. Show Disk Usage in Megabytes for the Register Directory: /dir

- Command: `hdfs dfs [generic options] -du [-s] [-h] <path> ...`
- Note: The -h, which is marked in Blue gives you a readable output of size, i.e., Gigabytes.

5. Modifying the replication factor for a file

- Command: `hadoop fs -setrep -w 1 /root/journaldev_bigdata/derby.log`
- Note: It is for replication factors, which count by a file, which can be replicated in each Hadoop cluster.

6. copyFromLocal

- Command: `hadoop fs -copyFromLocal derby.log /root/journaldev_bigdata`
- Note: This command is for copy of a file from Local file System to Hadoop FS

7.-rm -r

- Command: `hadoop fs -rm -r /root/journaldev_bigdata`
- Note: With the help of rm-r command, we can remove an entire HDFS directory

8. Expunge

- Command: `hadoop fs -expunge`
- Note: This expunge performs fragments empty.

9. fs -du

- Command: `hadoop fs -du /root/journaldev_bigdata/`
- Note: This command helps to disk usage of files under HDFS in a directory.

10.mkdir

- Command: `hadoop fs -mkdir /root/journaldev_bigdata`
- Note: This command is used for checking the health of the files.

11.text

- Command: `hadoop fs -text <src>`
- Note: This command is used to visualise the “sample zip” file in text format.

12. Stat

- Command: `hadoop fs -stat [format] <path>`
- Note: This stat command is used to print the information about the 'test' file present in the directory.

13. chmod : (Hadoop chmod Command Usage)

- Command: `hadoop fs -chmod [-R] <mode> <path>`
- Note: This command is used for changing the file permission on "testfile".
- Note: This command generates a list of available files and subdirectories under default directory.

14. appendToFile

- Command: `hadoop fs -appendToFile <localsrc> <dest>`
- Note: This command can be used for appending the localfile1, localfile2 instantly in the local filesystem into the file specified as 'appendfile' in the catalog.

15. Checksum

- Command: `hadoop fs -checksum <src>`
- Note: This is the shell command which returns the checksum information.

RESULT: The exploration of Hadoop daemons and HDFS commands have been accomplished successfully.

Program 3	Write a program to implement a word count program using MapReduce
Date-	

Step 1:

Create a file with the name word_count_data.txt and add some data to it.

```
cat >word_count_data.txt          # cat is used to type the content of the file
```

```
cat word_count_data.txt          # cat is used to see the content of the file
```

Step 2:

Create a **mapper.py** file that implements the mapper logic. It will read the data from STDIN and will split the lines into words, and will generate an output of each word with its individual count.

```
cat word_count_data.txt
```

go to the file located and type the file

```
#!/usr/bin/env python3
```

```
# import sys because we need to read and write data to STDIN and STDOUT
```

```
import sys
```

```
# reading entire line from STDIN (standard input)
```

```
for line in sys.stdin:
```

```
    # to remove leading and trailing whitespace
```

```
    line = line.strip()
```

```
    # split the line into words
```

```
    words = line.split()
```

```
    # we are looping over the words array and printing the word
```

```
    # with the count of 1 to the STDOUT
```

```
    for word in words:
```

```
        # write the results to STDOUT (standard output);
```

```
# what we output here will be the input for the
# Reduce step, i.e. the input for reducer.py
print ('%s\t%s' % (word, 1) )
```

Step 3:

Create a **reducer.py** file that implements the reducer logic. It will read the output of mapper.py from STDIN(standard input) and will aggregate the occurrence of each word and will write the final output to STDOUT.

Create it in the same location as mapper.py

Reducer.py

```
#!/usr/bin/env python3
```

```
from operator import itemgetter
import sys
```

```
current_word = None
```

```
current_count = 0
```

```
word = None
```

```
# read the entire line from STDIN
```

```
for line in sys.stdin:
```

```
    # remove leading and trailing whitespace
```

```
    line = line.strip()
```

```
    # splitting the data on the basis of tab we have provided in mapper.py
```

```
    word, count = line.split('\t', 1)
```

```
    # convert count (currently a string) to int
```

```
    try:
```

```
        count = int(count)
```

```
    except ValueError:
```

```
# count was not a number, so silently
# ignore/discard this line
continue
```

```
# this IF-switch only works because Hadoop sorts map output
# by key (here: word) before it is passed to the reducer
if current_word == word:
    current_count += count
else:
    if current_word:
        # write result to STDOUT
        print '%s\t%s' % (current_word, current_count)
    current_count = count
    current_word = word
```

do not forget to output the last word if needed!

```
if current_word == word:
    print ('%s\t%s' % (current_word, current_count) )
```

Step 4:

Now let's start all our Hadoop daemons with the below command.

```
start-dfs.sh
start-yarn.sh
```

Now make a directory **word_count_in_python** in our HDFS in the root directory that will store our **word_count_data.txt** file with the below command.

```
hdfs dfs -mkdir /word_count_in_python
```

Let's give executable permission to our **mapper.py** and **reducer.py** with the help of below command.

```
chmod 777 mapper.py reducer
```

Syntax to copy a file from your local file system to the HDFS is given below:

```
hdfs dfs -copyFromLocal /path 1 /path 2...../path n /destination
```

Step 5

Now download the latest **hadoop-streaming jar** file from this Link- <https://jar-download.com/artifacts/org.apache.hadoop/hadoop-streaming/2.7.3/source-code>.

Then place, this Hadoop,-streaming jar file to a place from you can easily access it. In my case, I am placing it to **/Documents** folder where **mapper.py** and **reducer.py** file is present.

Now let's run our python files with the help of the Hadoop streaming utility as shown below.

```
hadoop jar /nmit /home/Documents/hadoop-streaming-2.7.3.jar \
```

```
> -input /word_count_in_python/word_count_data.txt \
```

```
> -output /word_count_in_python/output \
```

```
> -mapper /home/dikshant/Documents/mapper.py \
```

```
> -reducer /home/dikshant/Documents/reducer.py
```

Output	
GeeksforGeeks	1
Hello	2
I	2
Intern	1
am	2
an	1

RESULT: A program to implement a word count program using MapReduce has been completed successfully.

Program 4	Write a program to analyze stock dataset using MapReduce.
Date-	

Step 1:

log in to nmit

Step2:

launch anaconda.

In terminal type – anaconda-navigator

Step3:

create an environment with python version 3.8and install jupyter

Step4:

launch jupyter and install yfinance and MRJob

In the notebook type these commands

! pip install yfinance

! pip install MRJob

Go back to anaconda navigator and check if hdfs3 and lib hdfs is downloaded if not downloaded download it

Step 5:

Log to Hadoop user and repeat step 2. Use the same environment created in nmit

Launch jupyter

Step 6:

Type the code

```
import yfinance as yf
```

```
data = yf.download("AAPL IBM", start="2009-01-01", end="2019-12-31")
```

```
%%bash
```

```
$HADOOP_HOME/sbin/start-dfs.sh
```

```
$HADOOP_HOME/sbin/start-yarn.sh
```

```
from hdfs3 import HDFFileSystem
```

```
hdfs = HDFFileSystem(host='localhost', port=9000)
```

```
with hdfs.open('/AAPL_IBM_open.csv', 'wb') as f:
```

```
    data['Open'].to_csv(f,header=True)
```

```
%%file stock_analysis.py
```

```
from mrjob.job import MRJob
```

```
import re
```

```
import sys
```

```
class StockAnalysis(MRJob):
```

```
    def mapper(self, key, value):
```

```
        date, apple_open, samsung_open = value.split(',')
```

```
        #print(value, file=sys.stderr)
```

```
        year = date[:4]
```

```
        month = date[5:7]
```

```
        if (month=='10' or month=='11' or month=='12'):
```

```
            apple_key = 'apple_%s' % year
```

```
            samsung_key = 'samsung_%s' % year
```

```
            yield(apple_key, float(apple_open))
```

```
            yield(samsung_key, float(samsung_open))
```

```
    def reducer(self, key, values):
```

```
        yield(key, max(values))
```

```
if __name__ == '__main__':  
    StockAnalysis.run()
```

```
!python stock_analysis.py -r hadoop hdfs:///AAPL_IBM_open.csv
```

Output

```
"apple_2009" 7.611785888671875  
"apple_2010" 11.650713920593262  
"apple_2011" 15.062856674194336  
"apple_2012" 23.97321319580078  
"apple_2013" 20.451786041259766  
"apple_2014" 29.8174991607666  
"apple_2015" 30.782499313354492  
"apple_2016" 29.545000076293945  
"apple_2017" 43.77750015258789  
"apple_2018" 57.69499969482422  
"apple_2019" 72.77999877929688  
"samsung_2009" 132.41000366210938  
"samsung_2010" 146.72999572753906  
"samsung_2011" 193.63999938964844  
"samsung_2012" 211.14999389648438  
"samsung_2013" 186.49000549316406  
"samsung_2014" 189.91000366210938  
"samsung_2015" 152.4600067138672  
"samsung_2016" 168.97000122070312  
"samsung_2017" 162.0500030517578  
"samsung_2018" 154.0  
"samsung_2019" 145.58999633789062
```

RESULT: A program to analyze stock dataset using MapReduce has been completed successfully.

Program 5	Write a program to analyze temperature dataset using map Reduce
Date-	

Step 1:

store the dataset which to be analyzed

link for dataset- https://www.ncei.noaa.gov/pub/data/uscrn/products/daily01/2002/CRND0103-2002-RI_Kingston_1_NW.txt name it as temp_data.txt

Step 2:

Create mapper.py

```
#!/usr/bin/env python3
```

```
import sys
```

```
# input comes from STDIN (standard input)
```

```
# the mapper will get daily max temperature and group it by month. so output will be
(month,dailymax_temperature)
```

```
for line in sys.stdin:
```

```
    # remove leading and trailing whitespace
```

```
    line = line.strip()
```

```
    # split the line into words
```

```
    words = line.split()
```

```
    #See the README hosted on the weather website which help us understand how each position
represents a column
```

```
    month = line[10:12]
```

```
    daily_max = line[38:45]
```

```
    daily_max = daily_max.strip()
```

```
    # increase counters
```

```
    for word in words:
```

```
        # write the results to STDOUT (standard output);
```

```
        # what we output here will be go through the shuffle process and then
```

```
# be the input for the Reduce step, i.e. the input for reducer.py
#
# tab-delimited; month and daily max temperature as output
print ('%s\t%s' % (month ,daily_max))
```

Step 3 :

create reducer.py

```
#!/usr/bin/env python3
```

```
from operator import itemgetter
```

```
import sys
```

```
#reducer will get the input from stdid which will be a collection of key, value(Key=month , value= daily
max temperature)
```

```
#reducer logic: will get all the daily max temperature for a month and find max temperature for the
month
```

```
#shuffle will ensure that key are sorted(month)
```

```
current_month = None
```

```
current_max = 0
```

```
month = None
```

```
# input comes from STDIN
```

```
for line in sys.stdin:
```

```
    # remove leading and trailing whitespace
```

```
    line = line.strip()
```

```
    # parse the input we got from mapper.py
```

```
    month, daily_max = line.split('\t', 1)
```

```
    # convert daily_max (currently a string) to float
```

```
    try:
```

```
        daily_max = float(daily_max)
```

```
    except ValueError:
```

```
        # daily_max was not a number, so silently
```

```
# ignore/discard this line
continue
```

```
# this IF-switch only works because Hadoop shuffle process sorts map output
# by key (here: month) before it is passed to the reducer
```

```
if current_month == month:
    if daily_max > current_max:
        current_max = daily_max
    else:
        if current_month:
            # write result to STDOUT
            Print( '%s\t%s' % (current_month, current_max))
        current_max = daily_max
        current_month = month
```

```
# output of the last month
```

```
if current_month == month:
    print( '%s\t%s' % (current_month, current_max))
```

Step 4:

```
cat dataset.txt | python mapper.py | sort -k1,1 | python reducer.py
```

Step 5:

type this command in terminal - `chmod 777 mapper.py reducer.py`

Step 6:

Type the commands in terminal.

```
su - hadoop
```

```
start-dfs.sh
```

```
start-yarn.sh
```

Step 7

make a directory

```
hdfs dfs -mkdir /temp-python
```

Step 8

type hdfs command in this format

```
hdfs dfs -copyFromLocal /path 1 /path 2..... /path n /dest
```

Step 9

Now download the latest **hadoop-streaming jar** file from this Link- <https://jar-download.com/artifacts/org.apache.hadoop/hadoop-streaming/2.7.3/source-code>.

Then place, this Hadoop,-streaming jar file to a place from you can easily access it. In my case, I am placing it to **/Documents** folder where **mapper.py** and **reducer.py** file is present.

```
hadoop jar /nmit /home /Documents/hadoop-streaming-2.7.3.jar \
```

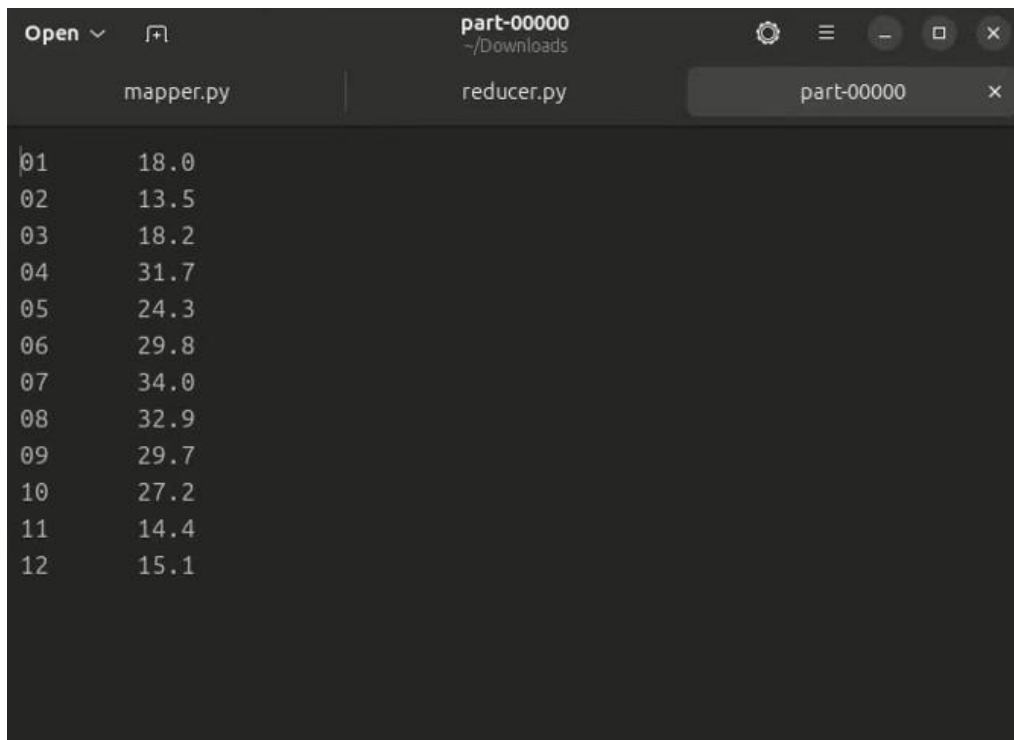
```
> -input /word_count_in_python/ temp_data.txt \
```

```
> -output /word_count_in_python/output \
```

```
> -mapper /home/dikshant/Documents/mapper.py \
```

```
> -reducer /home/dikshant/Documents/reducer.py
```

Output



```
01      18.0
02      13.5
03      18.2
04      31.7
05      24.3
06      29.8
07      34.0
08      32.9
09      29.7
10      27.2
11      14.4
12      15.1
```

RESULT: A program to analyze temperature dataset using map Reduce has been completed successfully.

Program 6	Working with hive commands to analyze bucketing and partitioning.
Date-	

PARTITIONING

Step 1:

Launch Hadoop and start yarn and hdfs

Step 2:

Launch hive

And create database userb

Syntax- create database userb

Step 3:

Create table allcountries

Syntax- create table all_countries (country String, state String, Enrollment String) row format delimited by ',';

Working with hive commands to analyze bucketing and partitioning.

Create a csv file to store all the details of the table

Cat > details.csv

"India", "karnataka", "2345"

"USA", "Kansas", "453"

"Japan", "Tokyo", "345"

Step 5:

Push the details csv into hive

Load data local inpath 'home/nmit/details.csv' into table allcountries

Step 6 :

Before partition

Set.hive.dynamic.partition.mode=nonstrict;

Step 7:

Create partition table

Create table countries_part (countries String, Enrollment String) partitioned by (country string);

Step 8 :

INSERT OVERWRITE TABLE countries_part PARTITION(countries) select state,enrollment,countries from all_countries;

Partition complete
to view the partition -
select * from countries_part

BUCKETING

Step 1:

in hive

create table tab1(eid int, name String, dept String, doj int)row format delimited by ',';

Step 2:

Create a csv file to store all the details of the table

Cat > details.csv

```
1,"Rahul","AIDS",2012
2,"Rotem","AIDS",2012
3,"Ram","AIDS",2014
4,"Rohan","AIDS",2014
```

Step 3:

create table sample sample bucket

create table sameple_bucket(eid int, name String, dept String, doj int) clustered by (doj) into 2 buckets
row format delimited fields terminated by ',';

Step 4 :

from tab1 insert overwrite table sameple_bucket select eid,name, dept,doj;

Step 5:

go to local host website with appropriate port number to view your results

RESULT: The successful completion of tasks involving Hive commands for analyzing bucketing and partitioning has been achieved.

Program 7	Working with hbase commands to analyze dataset.
Date-	

Step 1

launch hbase

Step 2

create a database userb

Step 3

use database userb

Step 4

create a table

Syntax: create '<name_space:table_name>', '<column_family>
create 'emp','office';

Step 5

```
put 'emp', '3', 'office:salary', '10000'
put 'emp', '3', 'office:name', 'Jeff'
put 'emp', '3', 'office:salary', '20000'
put 'emp', '1', 'office:name', 'Scott'
put 'emp', '2', 'office:name', 'Mark'
put 'emp', '2', 'office:gender', 'M'
```

Step 6

```
get '<namespace>:<table_name>', '<row_key>', '<column_key>'
get 'emp','1'
get 'emp','3',{COLUMN => ['office:salary','office:name']}
```

RESULT: The successful completion of tasks involving HBase commands for analyzing the dataset has been accomplished.

Program 8	Installation and configuration of spark, word count analysis using scala
Date-	

Step 1:

Install Java

Install java8 using the below command.

```
sudo apt-get install oracle-java8-installer
```

The above command creates a java-8-oracle Directory in /usr/lib/jvm/ directory in your machine. Now we need to configure the JAVA_HOME path in .bashrc file. .bashrc file executes whenever we open the terminal. Configure JAVA_HOME and PATH in .bashrc file and save. To edit/modify .bashrc file, use the below command.

```
vi .bashrc
```

Then press i(for insert) -> then Enter the below the line at the bottom of the file.

```
export JAVA_HOME= /usr/lib/jvm/java-8-oracle/
```

```
export PATH=$PATH:$JAVA_HOME/bin
```

Then Press Esc -> wq! (For save the changes) -> Enter. Now, test whether Java is installed properly or not by checking the version of Java. The below command should show the java version.

```
java -version
```

Step 2:

Installation of Spark in the System

Go to the official download page of Apache Spark below and choose the latest release. For the package type, choose 'Pre-built for Apache Hadoop'.

Step 3:

Creating Spark Directory

Create a directory called spark under /usr/ directory. Use the below command to create a spark directory

```
sudo mkdir /usr/spark
```

The above command asks password to create a spark directory under the /usr directory; you can give the password. Then check spark directory is created or not in the /usr directory using the below command

```
ll /usr/
```

Go to /usr/spark directory. Use the below command to go spark directory.

```
cd /usr/spark
```

Download spark2.3.3 in the spark directory.

Step 4: Extract Spark file

Then extract spark-2.4.0-bin-hadoop2.7.tgz using the below command.

```
sudo tar xvfz spark-2.4.0-bin-hadoop2.7
```

Now spark-2.4.0-bin-hadoop2.7.tgz file is extracted as spark-2.4.0-bin-hadoop2.7. Check whether it extracted or not using ll command.

Step 5:

Configuration

Configure the SPARK_HOME path in the .bashrc file by following the below steps. Go to the home directory using the below command.

```
cd ~
```

Open the .bashrc file using the below command

```
vi .bashrc
```

Now we will configure SPARK_HOME and PATH. Press i for insert the enter SPARK_HOME and PATH like below

```
SPARK_HOME=/usr/spark/spark-2.4.0-bin-hadoop2.7
```

```
PATH=$PATH:$SPARK_HOME/bin
```

Then save and exit by entering the below commands. Press Esc -> wq! -> Enter

Step 6:

Test Installation

Now we can verify whether the spark is successfully installed in our Ubuntu Machine. To verify, use the below command, then enter.

```
spark-shell
```

Word Count in Scala

Step 1

In spark directory create a text file called 'sparkdata.txt' and type following content:

"I am a good person, He is a good person, She is a good person"

Step 2

Now open the spark-shell using the command:

```
$ spark-shell
```

Step 3

Load the txtfile

```
scala> val data=sc.textFile("sparkdata.txt")
```

```
scala> data.collect;
```

Step 4

Work with the data

```
scala> val splitdata = data.flatMap(line => line.split(" "));
```

```
scala> splitdata.collect;
```

Now for the mapper and reducer:

```
scala> val mapdata = splitdata.map(word => (word,1));
```

```
scala> val reducedata = mapdata.reduceByKey(_+_);
```

Step 5

```
scala> reducedata.collect;
```

RESULT: Installation and configuration of spark, word count analysis using scala has been completed successfully.

Program 9	Implementing on Saving RDD – Lazy Operation – Spark Jobs- Spark ML library
Date-	

Step 1

Open Spark shell:

```
$shell-spark
```

Step 2

Create a dataset of values from 1 to 100000

```
val data = (1 to 100000).toList
```

Step 3

Create 4 partitions

```
val rdd = sc.parallelize(data,4) println("Number of partitions is "+rdd.getNumPartitions)
```

Step 4

Perform Fundamental Transformations

```
//Adding 5 to each value in rdd
```

```
val rdd2 = rdd.map(x => x+5)
```

```
//rdd2 object
```

```
println(rdd2)
```

```
//getting rdd lineage
```

```
rdd2.toDebugString
```

```
//Adding 5 to each value in rdd
```

```
val rdd3 = rdd2.map(x => x+20)
```

```
//rdd2 object
```

```
println(rdd3)
```

```
//getting rdd lineage
```

```
rdd3.toDebugString
```

```
rdd3.collect
```

RESULT: The successful implementation of saving RDD, including lazy operations and Spark jobs, within the Spark ML library has been accomplished.

Program 10	Implement K-means Clustering algorithm using Map Reduce
Date-	

Step 1

First go to apache spark directory and create a file named 'mean.scala'

Step 2

Create a file named 'kmeans_data.txt' and add this:

```
0 1:0.0 2:0.0 3:0.0
1 1:0.1 2:0.1 3:0.1
2 1:0.2 2:0.2 3:0.2
3 1:9.0 2:9.0 3:9.0
4 1:9.1 2:9.1 3:9.1
5 1:9.2 2:9.2 3:9.2
```

Step 3

Type this into the means.scala file:

```
import org.apache.spark.ml.clustering.KMeans
import org.apache.spark.ml.evaluation.ClusteringEvaluator

// Loads data.
val dataset = spark.read.format("libsvm").load("kmeans_data.txt")

// Trains a k-means model.
val kmeans = new KMeans().setK(2).setSeed(1L)
val model = kmeans.fit(dataset)

// Make predictions
val predictions = model.transform(dataset)

// Evaluate clustering by computing Silhouette score
val evaluator = new ClusteringEvaluator()
val silhouette = evaluator.evaluate(predictions)
println(s"Silhouette with squared euclidean distance = $silhouette")
```

```
// Shows the result.  
println("Cluster Centers: ")  
model.clusterCenters.foreach(println)
```

Step 4

Go back to the terminal and type

```
spark-shell -i mean.scala
```

RESULT: The successful implementation of the K-means Clustering algorithm using MapReduce has been achieved.

Internal Evaluation

Date:

Name of the program obtained:

Sl.no	Particulars	Max Marks	Obtained Marks
1.	Observation Book	10	
2.	Record Book	10	
3.	Compilation of Program	12	
4.	Execution of Program	13	
5.	Viva	05	
	Total	50	

Total marks secured:

Signature of Faculty

