Experiment No. 01

Aim: To install Hadoop on linux operating syatem.

Theory: Apache Hadoop is an open source framework that is used to efficiently store and process large datasets ranging in size from gigabytes to petabytes of data. Instead of using one large computer to store and process the data, Hadoop allows clustering multiple computers to analyze massive datasets in parallel more quickly.
Hadoop consists of four main modules:

- Hadoop Distributed File System (HDFS) – A distributed file system that runs on standard or low-end hardware. HDFS provides better data throughput than traditional file systems, in addition to high fault tolerance and native support of large datasets.
- Yet Another Resource Negotiator (YARN) – Manages and monitors cluster nodes and resource usage. It schedules jobs and tasks.
- MapReduce – A framework that helps programs do the parallel computation on data. The map task takes input data and converts it into a dataset that can be computed in key value pairs. The output of the map task is consumed by reduce tasks to aggregate output and provide the desired result.
- Hadoop Common – Provides common Java libraries that can be used across all modules.

Steps to install Hadoop

Step 1: Install Java

sudo apt update
sudo apt install openjdk-11-jdk

Step 2: Create a Hadoop User

sudo adduser Hadoop
su – hadoop

Step 3: Configure SSH Key-based Authentication

ssh-keygen -t rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 640 ~/.ssh/authorized_keys
ssh localhost

Step 4: Installing Hadoop

wget https://downloads.apache.org/hadoop/common/hadoop-3.3.0/hadoop-3.3.0.tar.gz
tar -xvzf hadoop-3.3.0.tar.gz
mv hadoop-3.3.0 hadoop
nano ~/.bashrc
Append the below lines to file.
export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
export HADOOP_HOME=/home/hadoop/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export HADOOP_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/native"

Step 5 - Configuring Hadoop

First, you will need to create the namenode and datanode directories inside Hadoop home directory:

mkdir -p ~/hadoopdata/hdfs/namenode
mkdir -p ~/hadoopdata/hdfs/datanode
nano $HADOOP_HOME/etc/hadoop/core-site.xml
Change the following name as per your system hostname:

XHTML

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://hadoop.tecadmin.com:9000</value>
    </property>
</configuration>
```

```
<configuration>
    <property>
        <name>fs.defaultFS</name>
        <value>hdfs://hadoop.tecadmin.com:9000</value>
    </property>
</configuration>
```

Save and close the file. Then, edit the hdfs-site.xml file:

nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml
Change the NameNode and DataNode directory path as shown below:

XHTML

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>

    <property>
        <name>dfs.name.dir</name>
        <value>file:///home/hadoop/hadoopdata/hdfs/namenode</value>
    </property>

    <property>
        <name>dfs.data.dir</name>
        <value>file:///home/hadoop/hadoopdata/hdfs/datanode</value>
    </property>
</configuration>
```

```
<configuration>
    <property>
        <name>dfs.replication</name>
        <value>1</value>
    </property>
```

```
        <property>
            <name>dfs.name.dir</name>
            <value>file:///home/hadoop/hadoopdata/hdfs/namenode</value>
        </property>

        <property>
            <name>dfs.data.dir</name>
            <value>file:///home/hadoop/hadoopdata/hdfs/datanode</value>
        </property>
    </configuration>
```

Save and close the file. Then, edit the mapred-site.xml file:

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

Make the following changes:

XHTML

```
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
<configuration>
    <property>
        <name>mapreduce.framework.name</name>
        <value>yarn</value>
    </property>
</configuration>
```

Save and close the file. Then, edit the yarn-site.xml file:

nano $HADOOP_HOME/etc/hadoop/yarn-site.xml

Make the following changes:

XHTML

```
<configuration>
    <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
    </property>
</configuration>
<configuration>
    <property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
</configuration>
```

Step 6 - Start Hadoop Cluster

Run the following command to format the hadoop Namenode:

hdfs namenode -format

start-dfs.sh

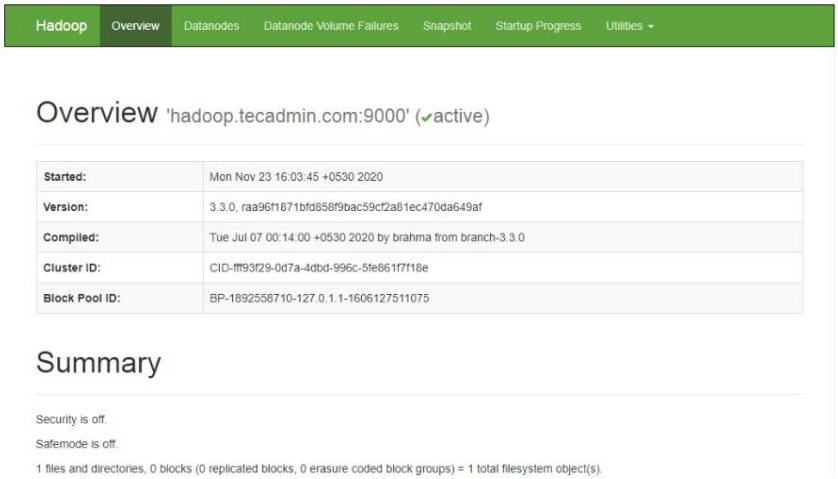
start-yarn.sh

jps

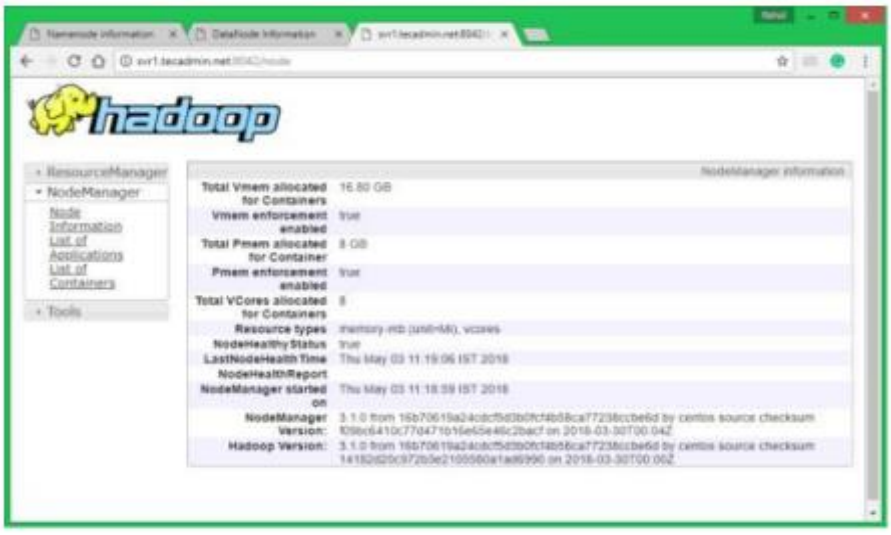
Step 7 - Access Hadoop Namenode and Resource Manager
Open any browser Hadoop localhost by using below URL
http://localhost:9870

Result:

1. Hadoop NameNode started on port 9870 default. Access your server on port 9870 in your favorite web browser.
   http://localhost:9870/



2. Now access port 8042 for getting the information about the cluster and all applications http:// localhost:8042/

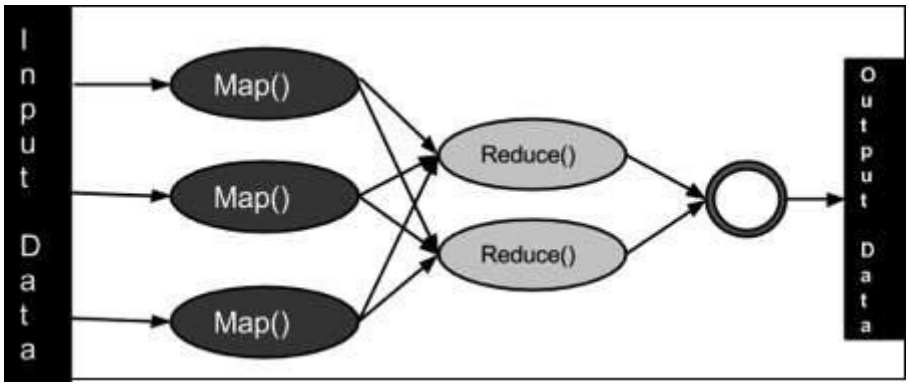


Conclusion:
As the technology is evolving data generation rate is also increasing day by day. Storing and processing this massive volume of data is not possible by using traditional systems like RDBMS. This problem can be overcome by using Hadoop. Hadoop is framework used to store and process big data.
In this experiment Hadoop framework is installed and studies it's working.

Experiment No. 02

Aim: To build Hadoop MapReduce application for counting frequency of word/phrase in simple text file.

Theory: MapReduce is a processing technique and a program model for distributed computing based on java. The MapReduce algorithm contains two important tasks, namely Map and Reduce. Map takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key/value pairs). Secondly, reduce task, which takes the output from a map as an input and combines those data tuples into a smaller set of tuples. As the sequence of the name MapReduce implies, the reduce task is always performed after the map job.



The major advantage of MapReduce is that it is easy to scale data processing over multiple computing nodes. Under the MapReduce model, the data processing primitives are called mappers and reducers. Decomposing a data processing application into mappers and reducers is sometimes nontrivial. But, once we write an application in the MapReduce form, scaling the application to run over hundreds, thousands, or even tens of thousands of machines in a cluster is merely a configuration change. This simple scalability is what has attracted many programmers to use the MapReduce model.

Steps to run WordCount Application in Eclipse

Step-1: Download eclipse if you don't have.

Step-2: Open Eclipse and Make Java Project.
In eclipse Click on File menu-> new -> Java Project. Write there your project name. Here is WordCount. Make sure Java version must be 1.6 and above. Click on Finish.

Step-3: Make Java class File and write a code.
Click on WordCount project. There will be 'src' folder. Right click on 'src' folder -> New -> Class. Write Class file name. Here is Wordcount. Click on Finish.
Copy and Paste below code in Wordcount.java. Save it.
You will get lots of error but don't panic. It is because of requirement of external library of hadoop which is required to run mapreduce program.

```
import java.io.IOException;
import java.util.StringTokenizer;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
```

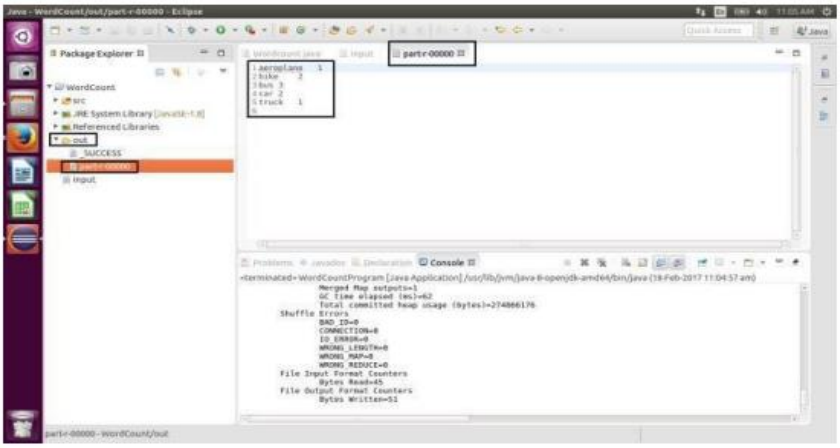
```
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.fs.Path;
public class WordCount
{
		public static class Map extends Mapper<LongWritable,Text,Text,IntWritable>
		{
		public void map(LongWritable key, Text value,Context context) throws
		IOException,InterruptedException{
		String line = value.toString();
		StringTokenizer tokenizer = new StringTokenizer(line);
		while (tokenizer.hasMoreTokens())
			{
			value.set(tokenizer.nextToken());
			context.write(value, new IntWritable(1));
			}
		}
	}

	public static class Reduce extends Reducer<Text,IntWritable,Text,IntWritable>
	{
	public void reduce(Text key, Iterable<IntWritable> values,Context context)
	throws IOException,InterruptedException {
	int sum=0;
	for(IntWritable x: values)
		{
		sum+=x.get();
		}
	context.write(key, new IntWritable(sum));
	}
	}
	public static void main(String[] args) throws Exception
	{
	Configuration conf= new Configuration();
	Job job = new Job(conf,"My Word Count Program");
	job.setJarByClass(WordCount.class);
	job.setMapperClass(Map.class);
	job.setReducerClass(Reduce.class);
	job.setOutputKeyClass(Text.class);
	job.setOutputValueClass(IntWritable.class);
	job.setInputFormatClass(TextInputFormat.class);
	job.setOutputFormatClass(TextOutputFormat.class);
	Path outputPath = new Path(args[1]);
	FileInputFormat.addInputPath(job, new Path(args[0]));
	FileOutputFormat.setOutputPath(job, new Path(args[1]));
	outputPath.getFileSystem(conf).delete(outputPath);
	System.exit(job.waitForCompletion(true) ? 0 : 1);
	}
}
```

Step-4
Add external libraries from hadoop.
Right click on WordCount Project -> Build Path -> Configure Build Path -> Click on Libraries -> click on 'Add External Jars..' button

Step 5
Running Mapreduce Code.
5.1 Make input file for WordCount Project.
Right Click on WordCount project-> new -> File. Write File name and click on ok. You can copy and paste below contains into your input file.
car bus bike
bike bus aeroplane
truck car bus
5.2 Right click on WordCount Project -> click on Run As. -> click on Run Configuration…
Make new configuration by clicking on 'new launch configuration'.
Set Configuration
Name, Project Name and Class file name.

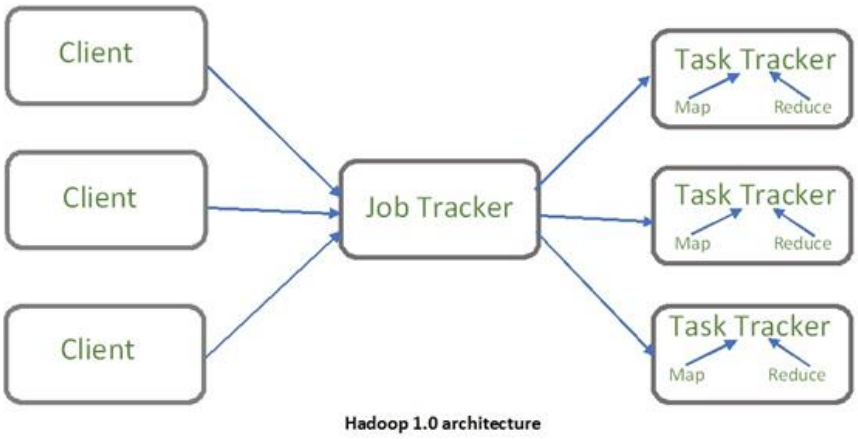
Result:
Output of WordCount Application and output logs in console.
Refresh WordCount Project. Right Click on project -> click on Refresh. You can find 'out' directory in project explorer. Open 'out' directory. There will be 'part-r-00000' file.
Double click to open it.



Experiment No. 03:

Aim: To study and demonstrate Hadoop YARN administration command and user commands.

Theory: YARN stands for "Yet Another Resource Negotiator ". It was introduced in Hadoop 2.0 to remove the bottleneck on Job Tracker which was present in Hadoop 1.0. YARN was described as a "Redesigned Resource Manager" at the time of its launching, but it has now evolved to be known as large-scale distributed operating system used for Big Data processing.



Hadoop 1.0 architecture

YARN also allows different data processing engines like graph processing, interactive processing, stream processing as well as batch processing to run and process data stored in HDFS (Hadoop Distributed File System) thus making the system much more efficient. Through its various components, it can dynamically allocate various resources and schedule the application processing. For large volume data processing, it is quite necessary to manage the available resources properly so that every application can leverage them.

YARN Features:

- YARN gained popularity because of the following features-
- Scalability: The scheduler in Resource manager of YARN architecture allows Hadoop to extend and manage thousands of nodes and clusters.
- Compatibility: YARN supports the existing map-reduce applications without disruptions thus making it compatible with Hadoop 1.0 as well.
- Cluster Utilization: Since YARN supports Dynamic utilization of cluster in Hadoop, which enables optimized Cluster Utilization.
- Multi-tenancy: It allows multiple engine access thus giving organizations a benefit of multi-tenancy.

YARN User commands:

Commands useful for users of a Hadoop cluster

1. **jar**
2. **application**
3. **node**
4. **logs**
5. **classpath**
6. **version**

Experiment No. 04

Aim: To install of R studio and demonstrate the following

1. R basic syntax
2. Exploring basic R data types
3. Drawing Pie chart, bar chart, and histogram
4. R array and vector

Theory: R is an open-source programming language that is widely used as a statistical software and data analysis tool. R generally comes with the Command-line interface. R is available across widely used platforms like Windows, Linux, and macOS. Also, the R programming language is the latest cutting-edge tool.

- R programming is used as a leading tool for machine learning, statistics, and data analysis. Objects, functions, and packages can easily be created by R.
- It's a platform-independent language. This means it can be applied to all operating system.
- It's an open-source free language. That means anyone can install it in any organization without purchasing a license.
- R programming language is not only a statistic package but also allows us to integrate with other languages (C, C++). Thus, you can easily interact with many data sources and statistical packages.
- The R programming language has a vast community of users and it's growing day by day.
- R is currently one of the most requested programming languages in the Data Science job market that makes it the hottest trend nowadays.

Features of R Programming Language

Statistical Features of R:

- Basic Statistics: The most common basic statistics terms are the mean, mode, and median. These are all known as "Measures of Central Tendency." So using the R language we can measure central tendency very easily.
- Static graphics: R is rich with facilities for creating and developing interesting static graphics. R contains functionality for many plot types including graphic maps, mosaic plots, biplots, and the list goes on.
- Probability distributions: Probability distributions play a vital role in statistics and by using R we can easily handle various types of probability distribution such as Binomial Distribution, Normal Distribution, Chi-squared Distribution and many more.
- Data analysis: It provides a large, coherent and integrated collection of tools for data analysis.

Programming Features of R:

- R Packages: One of the major features of R is it has a wide availability of libraries. R has CRAN(Comprehensive R Archive Network), which is a repository holding more than 10, 0000 packages.
- Distributed Computing: Distributed computing is a model in which components of a software system are shared among multiple computers to improve efficiency and performance. Two new packages *ddR* and *multidplyr* used for distributed programming in R were released in November 2015.

Implementation:

Installing Packages
The most common place to get packages from is CRAN. To install packages from CRAN you use install.packages("packagename"). For instance, if you want to install the ggplot2 package, which is a very popular visualization package you would type the following in the console:
install package from CRAN
install.packages("ggplot2")

Loading Packages
Once the package is downloaded to your computer you can access the functions and resources provided by the package in two different ways:
# load the package to use in the current R session
library(packagename)

Getting Help on Packages
For more direct help on packages that are installed on your computer you can use the help and vignette functions. Here we can get help on the ggplot2 package with the following:

help(package = "ggplot2") # provides details regarding contents of a package
vignette(package = "ggplot2") # list vignettes available for a specific package
vignette("ggplot2-specs") # view specific vignette
vignette() # view all vignettes on your computer

Assignment
The first operator you'll run into is the assignment operator. The assignment operator is used to assign a value. For instance, we can assign the value 3 to the variable x using the <- assignment operator.

x <- 3
Interestingly, R actually allows for five assignment operators:
# leftward assignment
x <- value
x = value
x <<- value
rightward assignment
value -> x
value ->> x

The original assignment operator in R was <- and has continued to be the preferred among R users. The = assignment operator was added in 2001 primarily because it is the accepted assignment operator in many other languages and beginners to R coming from other languages were so prone to use it. The operators <<- is normally only used in functions which we will not get into the details.

Evaluation
We can then evaluate the variable by simply typing x at the command line which will return the value of x. Note that prior to the value returned you'll see ## [1] in the command line. This simply implies that the output returned is the first output. Note that you can type any comments in your code by preceding the comment with the hash tag (#) symbol. Any values, symbols, and texts following # will not be evaluated.
evaluation
x

```
## [1] 3
```

Case Sensitivity

Lastly, note that R is a case sensitive programming language. Meaning all variables, functions, and objects must be called by their exact spelling:

```
x <- 1
y <- 3
z <- 4
x * y * z
## [1] 12
x * Y * z
## Error in eval(expr, envir, enclos): object 'Y' not found
```

Basic Arithmetic

At its most basic function R can be used as a calculator. When applying basic arithmetic, the PEMDAS order of operations applies: parentheses first followed by exponentiation, multiplication and division, and final addition and subtraction.

```
8 + 9 / 5 ^ 2
## [1] 8.36
8 + 9 / (5 ^ 2)
## [1] 8.36
8 + (9 / 5) ^ 2
## [1] 11.24
(8 + 9) / 5 ^ 2
## [1] 0.68
```

Implementing List and Vectors

Creating Vectors

The c() function can be used to create vectors of objects by concatenating things together.

```
> x <- c(0.5, 0.6) ## numeric
> x <- c(TRUE, FALSE) ## logical
> x <- c(T, F) ## logical
> x <- c("a", "b", "c") ## character
> x <- 9:29 ## integer
> x <- c(1+0i, 2+4i) ## complex
```

Numeric vector

```
x <- c(1,2,3,4,5,6)
```

The operator <– is equivalent to "=" sign.

Character vector

```
State <- c("DL", "MU", "NY", "DL", "NY", "MU")
```

Lists

```
> mylist <- list(x, y, z, gender, mydata)
> mylist
[[1]]
[1] 1 2 3 4 5

[[2]]
[1] 1 3 5 7 9

[[3]]
[1] 1 2 5 4 7

[[4]]
[1] "m" "f" "m" "m" "f"

[[5]]
  x y z gender
1 1 1 1      m
2 2 3 2      f
3 3 5 5      m
4 4 7 4      m
5 5 9 7      f
```

Drawing graphs (Visualization)
Data
data(Arthritis)

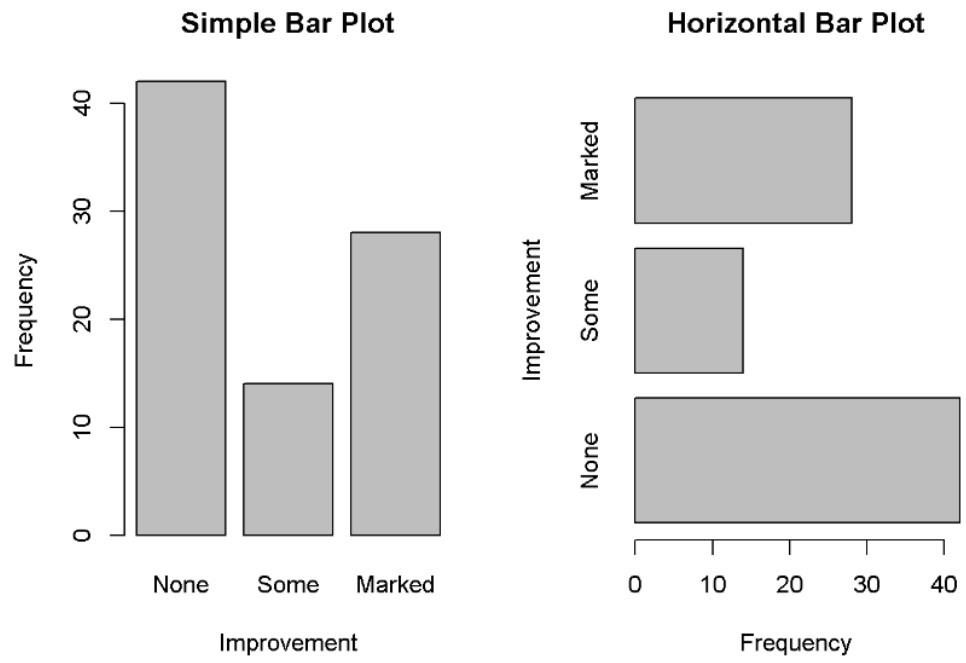
R Library Used
Basic R, ggplot2, vcd, plotrix

Bar Plots
A Bar plot displays the distribution/frequency of a categorical variable through vertical or horizontal bars.

```
par(mfrow=c(1,2)) # To display 2 graphs in one row
library(vcd)
counts <- table(Arthritis$Improved)
counts
```

```
##
##    None    Some Marked
##      42      14      28
```

```
# vertical barplot
barplot(counts,
        main="Simple Bar Plot",
        xlab="Improvement", ylab="Frequency")
# horizontal bar plot
barplot(counts,
        main="Horizontal Bar Plot",
        xlab="Frequency", ylab="Improvement",
        horiz=TRUE)
```



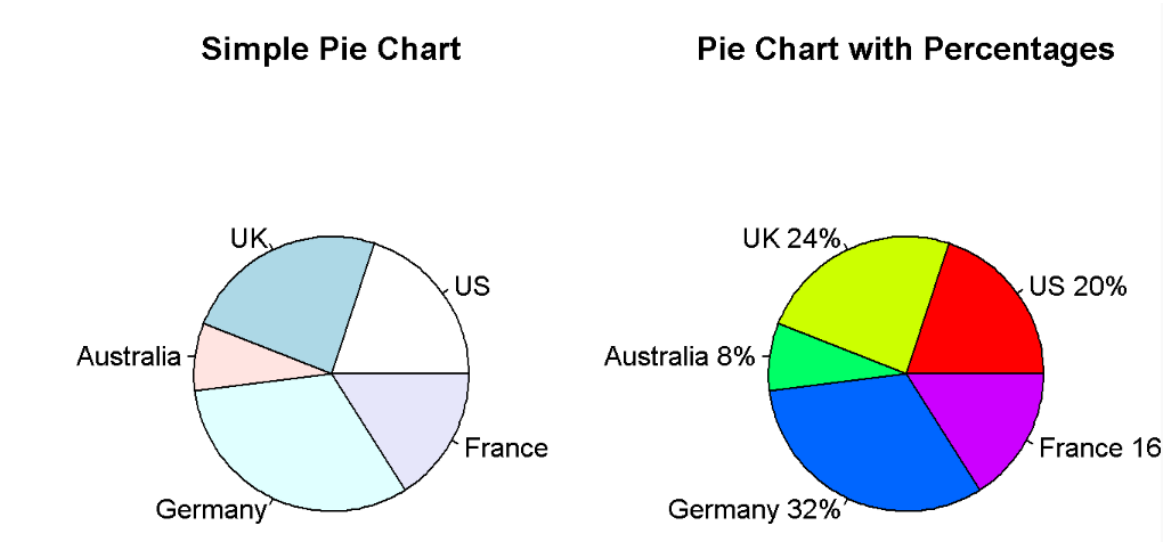
Pie charts

Pie charts are denigrated by most statisticians, including the authors of the R documentation. They recommend bar or dot plots over pie charts because people are able to judge length more accurately than volume.

```
par(mfrow=c(1,2)) # To display 2 graphs in one row
slices <- c(10, 12,4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")

pie(slices, labels = lbls,
    main="Simple Pie Chart")

pct <- round(slices/sum(slices)*100)
lbls <- paste(lbls, pct)
lbls <- paste(lbls,"%",sep="")
pie(slices,labels = lbls, col=rainbow(length(lbls)),
    main="Pie Chart with Percentages")
```



Result: Successfully installed R studio and demonstrated R basic syntax, drawn Bar chart, Histograms.

Experiment No. 05

Aim: To create, read and write R data sets and also create, read and write R tables.

Theory: R has a lot of built-in dataset. On the 4th panel is a tab called Packages. In Packages are several dataset packages such as boot, datasets and MASS. Click any of those packages and you will be taken to a page under the Help tab. The page shows you the datasets available. Click on a particular dataset to view the documentation for more information. To use any of the datasets, be sure there is a check mark beside the package.

Using "Import Dataset" in RStudio

RStudio makes importing datasets in a matter of clicks. First, go to the 4th panel's "Files" tab. Click "Upload" and upload the file you want. If the upload is successful, you should see the file appear under the "Files" tab.

Then go to the Environment (3rd) panel. You will see a dropdown arrow called "Import Dataset." The dropdown menu gives you a selection of what kind of data you want to import - whether it is a text file, excel file or csv file. Enter the file location, select the file and the file will appear in the Data preview area. If the file is correct, click "Import."

You should now see the file in the Environment panel under the Data heading. On the right of the imported filename, a statement on how many observations and how many variables are in the dataset will appear. In the Source (1st) panel, you will see a new tab with the imported filename. Click that tab to view the dataset.

Implementation:

Data Frame:

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items.

```
# Create the data frame.
emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15",
      "2015-03-27")),
   stringsAsFactors = FALSE
)
# Print the data frame.
print(emp.data)
```

When we execute the above code, it produces the following result –

```
  emp_id    emp_name     salary     start_date
1      1     Rick         623.30     2012-01-01
2      2     Dan          515.20     2013-09-23
3      3     Michelle     611.00     2014-11-15
4      4     Ryan         729.00     2014-05-11
5      5     Gary         843.25     2015-03-27
```

```
diabetes <- read.csv("r-intro-files/diabetes.csv")

class(diabetes)
```

```
## [1] "data.frame"
```

```
typeof(diabetes)
```

```
## [1] "list"
```

```
head(diabetes)
```

```
##   subject glyhb   location age gender height weight  frame
## 1   S1002  4.64 Buckingham  58 female     61    256  large
## 2   S1003  4.63 Buckingham  67   male     67    119  large
## 3   S1005  7.72 Buckingham  64   male     68    183 medium
## 4   S1008  4.81 Buckingham  34   male     71    190  large
## 5   S1011  4.84 Buckingham  30   male     69    191 medium
## 6   S1015  3.94 Buckingham  37   male     59    170 medium
```

```
colnames(diabetes)
```

```
## [1] "subject"  "glyhb"    "location" "age"      "gender"   "height"
## [7] "weight"   "frame"
```

```
ncol(diabetes)
```

```
## [1] 8
```

```
nrow(diabetes)
```

```
## [1] 354
```

Creating tables in R:

```
data <- data.frame(x1 = rep(LETTERS[1:2],       # Create example data frame
                            each = 4),
                   x2 = c(letters[1:3],
                          letters[2:5],
                          "b"))
data                                            # Print example data frame
```

Extract Subset of Table

```
tab5 <- tab1[tab1 > 1]                          # Extract table subset
tab5                                            # Print table subset
# b c
# 3 2
```

Reading table

You can use the read.table function to read in a file that contains tabular data into R. This function uses the following basic syntax:

```
df <- read.table(file='C:\\Users\\bob\\Desktop\\data.txt', header=FALSE, sep = "")
```

View the data frame:

```
#view data frame
print(df)

  var1 var2 var3
1    1    7    3
2    2    3    7
3    3    3    8
4    4    4    3
5    5    5    2
6    6    7    7
7    9    9    4
```

Result: Learnt how to create, read and write data sets in R and also learnt how to Create, read and write R tables.

Experiment No.6

Aim: Study the merging datasets, sorting data, putting data into shape, managing data using matrices managing data using data frames in R.

Theory:

Manipulating and processing data in R.:-

Data structures provide the way to represent data in data analytics. We can manipulate data in R for analysis and visualization.One of the most important aspects of computing with data in R is its ability to manipulate data and enable its subsequent analysis and visualization. Let us see few basic data structures in R:

Merging datasets:

Merge() Function in R.

Let us see the use of merge() function. The merge() function is used to combine data frames. Let us see this with an example: > merge(cold.states, large.states) Name Frost Area This is the command to create a data frame that consists of cold as well as large states. The merge() function allows four ways of combining data:

a. Natural join in R

To keep only rows that match from the data frames, specify the argument all=FALSE

b. Full outer join in R

To keep all rows from both data frames, specify all=TRUE

d. Right outer join in R.

To include all the rows of your data frame y and only those from x that match, specify all.y=TRUE

The merge()function takes a large number of arguments, as follows:

- x:A data frame
- y: A data frame
- by, by.x, by.y: Names of the columns common to both x and y. By default, it uses columns with common names between the two data frames.
- all, all.x, all.y: Logical values that specify the type of merge. The default value is all = FALSE.
-

```
names <- c('v1','v2','v3')
colnames(m1) <- names
colnames(m2) <- names
merge(m1,m2, by = names, all = TRUE)
```

sort function

The sort() function **sorts** the elements of a vector or a factor in increasing or decreasing order.

Sort(x, decreasing = FALSE, na.last = NA, . . .)

Syntax:

sort(x, decreasing, na.last)

Parameters:
x: Vector to be sorted
decreasing: Boolean value to sort in descending order
na.last: Boolean value to put NA at the end.

```
# R program to sort a vector

# Creating a vector

x <- c(7, 4, 3, 9, 1.2, -4, -5, -8, 6, NA)

# Calling sort() function

sort(x)
```

putting data into shape

The rgdal package offers the readOGR() function that allows to read shapefile using the following syntax.

- summary(my_spdf): tells you the max and min coordinates, the kind of projection in use.
- length(my_spdf): how many regions you have.
- head(my_spdf@data): the firs few rows of the data slot associated with the regions.

```
# Read this shape file with the rgdal library.
library(rgdal)
my_spdf <- readOGR(
  dsn= paste0(getwd(),"/DATA/world_shape_file/") ,
  layer="TM_WORLD_BORDERS_SIMPL-0.3",
  verbose=FALSE
)
```

Managing data using matrices.

A Matrix is created using the **matrix()** function

Syntax
The basic syntax for creating a matrix in R is –

matrix(data, nrow, ncol, byrow, dimnames)
Following is the description of the parameters used −

- data is the input vector which becomes the data elements of the matrix.
- nrow is the number of rows to be created.
- ncol is the number of columns to be created.
- byrow is a logical clue. If TRUE then the input vector elements are arranged by row.
- dimname is the names assigned to the rows and columns.

```
# Elements are arranged sequentially by row.
    M <- matrix(c(3:14), nrow = 4, byrow = TRUE)
    print(M)

  # Elements are arranged sequentially by column.
  N <- matrix(c(3:14), nrow = 4, byrow = FALSE)
  print(N)

  # Define the column and row names.
  rownames = c("row1", "row2", "row3", "row4")
  colnames = c("col1", "col2", "col3")

 P <- matrix(c(3:14), nrow = 4, byrow = TRUE, dimnames = list(rownames, colnames))
 print(P)
```

Managing data using data frames.

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items

Create Data Fram

```
    emp.data <- data.frame(
   emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),

   start_date = as.Date(c("2012-01-01", "2013-09-23", "2014-11-15", "2014-05-11",
     "2015-03-27")),
   stringsAsFactors = FALSE
  )
  # Print the data frame.
   print(emp.data)
```

Exercise:

1. Create a matrix taking a vector of numbers as input.

2. Extract specific column from a data frame using column name.

Conclusion: Students can able Manipulating and processing data in R - merging datasets, sorting data, putting data into shape, managing data using matrices managing data using data frames.

Experiment No .7

Problem Statement:.

Working with operators in Pig - FOREACH, ASSERT, FILTER, GROUP, ORDERBY, DISTINCT, JOIN, LIMIT, SAMPLE, SPLIT,FLATTEN.

Aim:

To understand the operators of pig **:** FOREACH, ASSERT, FILTER, GROUP, ORDERBY, DISTINCT, JOIN, LIMIT, SAMPLE, SPLIT,FLATTEN.

Thoery:

FOREACH:

The **FOREACH** operator is used to generate specified data transformations based on the column data.

Syntax

Given below is the syntax of **FOREACH** operator.

grunt> Relation_name2 = FOREACH Relatin_name1 GENERATE (required data);

student_details.txt

Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below.

001,Rajiv,Reddy,21,9848022337,Hyderabad
002,siddarth,Battacharya,22,9848022338,Kolkata

003,Rajesh,Khanna,22,9848022339,Delhi

004,Preethi,Agarwal,21,9848022330,Pune

005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar

006,Archana,Mishra,23,9848022335,Chennai

007,Komal,Nayak,24,9848022334,trivendram

008,Bharathi,Nambiayar,24,9848022333,Chennai

And we have loaded this file into Pig with the relation name **student_details** as shown below.

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt' USING PigStorage(',')
as (id:int, firstname:chararray, lastname:chararray,age:int, phone:chararray, city:chararray);
```

Let us now get the id, age, and city values of each student from the relation student_details and store it into another relation named foreach_data using the **foreach** operator.

```
grunt> foreach_data = FOREACH student_details GENERATE id,age,city;
```

ASSERT:

An Assert operator can be used for data validation. For example, the following script will fail if any value is a negative integer:

```
a = load 'something' as (a0: int, a1: int);
assert a by a0 > 0, 'a can't be negative for reasons';
```

FILTER

The **FILTER** operator is used to select the required tuples from a relation based on a condition.

Syntax:

```
grunt> Relation2_name = FILTER Relation1_name BY (condition);
```

Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below.

student_details.txt

001,Rajiv,Reddy,21,9848022337,Hyderabad
002,siddarth,Battacharya,22,9848022338,Kolkata
003,Rajesh,Khanna,22,9848022339,Delhi
004,Preethi,Agarwal,21,9848022330,Pune
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar
006,Archana,Mishra,23,9848022335,Chennai
007,Komal,Nayak,24,9848022334,trivendram
008,Bharathi,Nambiayar,24,9848022333,Chennai

And we have loaded this file into Pig with the relation name **student_details** as shown below

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt' USING PigStorage(',')
as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);
```

Let us now use the Filter operator to get the details of the students who belong to the city Chennai

```
filter_data = FILTER student_details BY city == 'Chennai';
```

GROUP:

The **GROUP** operator is used to group the data in one or more relations. It collects the data having the same key.

Syntax

Given below is the syntax of the **group** operator.

grunt> Group_data = GROUP Relation_name BY age;

Example

Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below.

student_details.txt

001,Rajiv,Reddy,21,9848022337,Hyderabad
002,siddarth,Battacharya,22,9848022338,Kolkata
003,Rajesh,Khanna,22,9848022339,Delhi
004,Preethi,Agarwal,21,9848022330,Pune
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar
006,Archana,Mishra,23,9848022335,Chennai
007,Komal,Nayak,24,9848022334,trivendram
008,Bharathi,Nambiayar,24,9848022333,Chennai

And we have loaded this file into Apache Pig with the relation name **student_details** as shown below.

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt' USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, age:int, phone:chararray, city:chararray);
```

Now, let us group the records/tuples in the relation by age as shown below.

grunt> group_data = GROUP student_details by age;

ORDERBY:

The **ORDER BY** operator is used to display the contents of a relation in a sorted order based on one or more fields.

Syntax

Given below is the syntax of the **ORDER BY** operator.

grunt> Relation_name2 = ORDER Relatin_name1 BY (ASC|DESC);

Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below.

student_details.txt

001,Rajiv,Reddy,21,9848022337,Hyderabad
002,siddarth,Battacharya,22,9848022338,Kolkata
003,Rajesh,Khanna,22,9848022339,Delhi
004,Preethi,Agarwal,21,9848022330,Pune
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar
006,Archana,Mishra,23,9848022335,Chennai
007,Komal,Nayak,24,9848022334,trivendram
008,Bharathi,Nambiayar,24,9848022333,Chennai

And we have loaded this file into Pig with the relation name student_details as shown below.

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt' USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray,age:int, phone:chararray, city:chararray);
```

Let us now sort the relation in a descending order based on the age of the student and store it into another relation named order_by_data using the ORDER BY operator as shown below.

```
grunt> order_by_data = ORDER student_details BY age DESC;
```

DISTINCT:

The **DISTINCT** operator is used to remove redundant (duplicate) tuples from a relation.

Syntax

Given below is the syntax of the DISTINCT operator.

```
grunt> Relation_name2 = DISTINCT Relatin_name1;
```

Example

Assume that we have a file named student_details.txt in the HDFS directory /pig_data/ as shown below.

student_details.txt

```
001,Rajiv,Reddy,9848022337,Hyderabad
002,siddarth,Battacharya,9848022338,Kolkata
002,siddarth,Battacharya,9848022338,Kolkata
003,Rajesh,Khanna,9848022339,Delhi
003,Rajesh,Khanna,9848022339,Delhi
004,Preethi,Agarwal,9848022330,Pune
005,Trupthi,Mohanthy,9848022336,Bhuwaneshwar
006,Archana,Mishra,9848022335,Chennai
006,Archana,Mishra,9848022335,Chennai
```

And we have loaded this file into Pig with the relation name student_details as shown below.

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt' USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray, phone:chararray, city:chararray);
```

Let us now remove the redundant (duplicate) tuples from the relation named student_details using the DISTINCT operator, and store it as another relation named distinct_data as shown below.

```
grunt> distinct_data = DISTINCT student_details;
```

JOIN:

The JOIN operator is used to combine records from two or more relations. While performing a join operation, we declare one (or a group of) tuple(s) from each relation, as keys. When these keys match, the two particular tuples are matched, else the records are dropped. Joins can be of the following types −

- Self-join
- Inner-join
- Outer-join − left join, right join, and full join

This chapter explains with examples how to use the join operator in Pig Latin. Assume that we have two files namely customers.txt and orders.txt in the /pig_data/ directory of HDFS as shown below.

customers.txt

1,Ramesh,32,Ahmedabad,2000.00

2,Khilan,25,Delhi,1500.00

3,kaushik,23,Kota,2000.00

4,Chaitali,25,Mumbai,6500.00

5,Hardik,27,Bhopal,8500.00

6,Komal,22,MP,4500.00

7,Muffy,24,Indore,10000.00

orders.txt

102,2009-10-08 00:00:00,3,3000

100,2009-10-08 00:00:00,3,1500

101,2009-11-20 00:00:00,2,1560

103,2008-05-20 00:00:00,4,2060

And we have loaded these two files into Pig with the relations customers and orders as shown below.

```
grunt> customers = LOAD 'hdfs://localhost:9000/pig_data/customers.txt' USING PigStorage(',')
   as (id:int, name:chararray, age:int, address:chararray, salary:int);


grunt> orders = LOAD 'hdfs://localhost:9000/pig_data/orders.txt' USING PigStorage(',')
   as (oid:int, date:chararray, customer_id:int, amount:int)
```

LIMIT:

The LIMIT operator is used to get a limited number of tuples from a relation.

Syntax

Given below is the syntax of the LIMIT operator.

```
grunt> Result = LIMIT Relation_name required number of tuples;
```

Assume that we have a file named **student_details.txt** in the HDFS directory /pig_data/ as shown below.

student_details.txt

001,Rajiv,Reddy,21,9848022337,Hyderabad
002,siddarth,Battacharya,22,9848022338,Kolkata
003,Rajesh,Khanna,22,9848022339,Delhi
004,Preethi,Agarwal,21,9848022330,Pune
005,Trupthi,Mohanthy,23,9848022336,Bhuwaneshwar
006,Archana,Mishra,23,9848022335,Chennai
007,Komal,Nayak,24,9848022334,trivendram
008,Bharathi,Nambiayar,24,9848022333,Chennai

And we have loaded this file into Pig with the relation name student_details as shown below.

```
grunt> student_details = LOAD 'hdfs://localhost:9000/pig_data/student_details.txt' USING PigStorage(',')
   as (id:int, firstname:chararray, lastname:chararray,age:int, phone:chararray, city:chararray);
```

Now, let's sort the relation in descending order based on the age of the student and store it into another relation named limit_data using the ORDER BY operator as shown below.

```
grunt> limit_data = LIMIT student_details 4;
```

SAMPLE:

Sample operator allows you to get the sample of data from your whole data-set i.e it returns the percentage of rows. It takes the value between 0 and 1. If it is 0.2, then it indicates 20% of the data.

grunt> emp_details = LOAD 'emp';

grunt> sample20 = SAMPLE emp_details BY 0.2;

SPLIT:

Pig Split operator is used to split a single relation into more than one relation depending upon the condition you will provide.

Let us suppose we have emp_details as one relation. We have to split the relation based on department number (dno). Sample data of emp_details as below:

mak,101,5000.0,500.0,10

ronning,102,6000.0,300.0,20

puru,103,6500.0,700.0,10

jetha,103,6500.0,700.0,30

grunt> SPLIT emp_details into emp_details1 IF dno=10, emp_details2 if (dno=20 OR dno=30);

grunt> DUMP emp_details1;

mak,101,5000.0,500.0,10

puru,103,6500.0,700.0,10

grunt> DUMP emp_details2;

ronning,102,6000.0,300.0,20

jetha,103,6500.0,700.0,30

FLATTEN:

Pig Flatten removes the level of nesting for the tuples as well as a bag. For Example: We have a tuple in the form of (1, (2,3)).
GENERATE expression $0 and flatten($1), will transform the tuple as (1,2,3).

When we un-nest a bag using flatten operator, then it creates tuples. For Example: we have bag as (1,{(2,3),(4,5)}).

GENERATE $0, flatten($1), then we create a tuple as (1,2,3), (1,4,5)

Exercise:

1. Explain self- join with example.
2. Explain Inner join with example.

Conclusion: Student able to understand Working with operators in Pig - FOREACH, ASSERT, FILTER, GROUP, ORDERBY, DISTINCT, JOIN, LIMIT, SAMPLE, SPLIT,FLATTEN.

Experiment No 8

Problem Statement:

Write and execute a Pig script

- Load data into a Pig relation without a schema
- Load data into a Pig relation with a schema
- Load data from a Hive table into a Pig relation

Aim: To Execute pig script with or without schema.

Theory:

How to load all the files from the given directory into Pig Relation without Schema?

```
grunt> a1 = load 'flightdelay' using PigStorage(',');
```

a1 is the **relation** with out schema.

Since /user/cloudera is a default **folder in hdfs**, we can use folder name directly.

using is the keyword which may or mayn't be case sensitive but **PigStorage** or any function should be Case sensitive otherwise it will throw an error.

all the statements in pig should end with **;(semicolon)**

How to load all the data into Pig Relation with schema?

```
grunt> A1 = load 'flightdelays' using PigStorage() AS (year:int,month:int,dayofmonth:int,ArrDely:int,Dest:chararray);
```

- though i have used a1 as relation name, i can still use A1 with capital letter both will act as a 2 different relations.
- now A1 relation will have schema with year, dayofmonth, arrdelay, dest…

Let us practice this using Pig Shell command prompt.

Without Schema details

```
grunt> a1 = load 'flightdelays/flight_delays1.csv' using PigStorage(',');
grunt>
```

With Schema details

```
grunt> a2 = load 'flightdelays/flight_delays1.csv' using PigStorage(',') as (year:int,month:int,dayofMonth:int,ArrDely:int,Dest:chararray);

grunt> describe a2;
a2: {year: int,month: int,dayofMonth: int,ArrDely: int,Dest: chararray}
```

Load data from a Hive table into a Pig relation:

Step 1: Load Data

Assume that we don't have any table in the hive. So, let's make it first. First start hive CLI, then create and load data into table "profits" which is under bdp schema. After executing below queries, verify that data is loaded successfully.
Use the below command to create a table:

```
1. CREATE SCHEMA IF NOT EXISTS bdp;
2. CREATE TABLE bdp.profits (product_id INT,profit BIGINT);
```

Use below command to insert data into table profits:

```
1. INSERT INTO TABLE bdp.profits VALUES
2. ('123','1365'),('124','3253'),('125','91522'),
3. ('123','51842'),('127','19616'),('128','2433'),
```

Verify data is loaded using below command.

```
select * from bdp.profits;
```

Step 2: Import table into pig

As we need to process this dataset using Pig, so let's go to grunt shell, use below command to enter into grunt shell, remember –useHCatalog is must as we need jars which are required to fetch data from a hive.

```
pig -useHCatalog;
```

Let's have one relation PROFITS in which we can load data from the hive table.

```
1. PROFITS = LOAD 'bdp.profits' USING org.apache.hive.hcatalog.pig.HCatLoader();
```

Step 3: Output

Enter below command to see whether data is loaded or not.

```
1. dump PROFITS
```

Exercise:

1. Calculate percentage in hive.

Conclusion: Students can understand how they can write pig script for Load data into a Pig relation without a schema,Load data into a Pig relation with a schema, Load data from a Hive table into a Pig relation.

Experiment No 9

Problem statement:

Installation and configuration of Apache Spark on Local Machine.

Aim: Install Apache Spark on Local Machine.

Theory:

Apache Spark is an open-source, general-purpose, multi-language analytics engine for large-scale data processing. It works on both single and multiple nodes by utilizing the RAM in clusters to perform fast data queries on large amounts of data. It offers batch data processing and real-time streaming, with support of high-level APIs in languages such as Python, SQL, Scala, Java or R. The framework offers in-memory technologies that allow it to store queries and data directly in the main memory of the cluster nodes.

Install Apache Spark:

install required packages.

$ sudo apt install curl mlocate git scala -y
Download Apache Spark. Find the latest release from the downloads page.

$ curl -O https://archive.apache.org/dist/spark/spark-3.2.0/spark-3.2.0-bin-hadoop3.2.tgz
Extract the Spark tarball.

$ sudo tar xvf spark-3.2.0-bin-hadoop3.2.tgz
Create an installation directory /opt/spark.

$ sudo mkdir /opt/spark
Move the extracted files to the installation directory.

$ sudo mv spark-3.2.0-bin-hadoop3.2/* /opt/spark
Change the permission of the directory.

$ sudo chmod -R 777 /opt/spark
Edit the bashrc configuration file to add Apache Spark installation directory to the system path.

$ sudo nano ~/.bashrc
Add the code below at the end of the file, save and exit the file:

export SPARK_HOME=/opt/spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
Save the changes to take effect.

$ source ~/.bashrc
Start the standalone master server.
$ start-master.sh

Conclusion: The student can able to install Apache spark on ubantu by following the step

Experiment No 10

Problem statement

Write an application to Read multiple text files into single RDD using Spark.

Aim: To read multiple text files into single RDD using Spark.

Theory:

we have three text files to read. We take the file paths of these three files as comma separated valued in a single string literal. Then using textFile() method, we can read the content of all these three text files into a single RDD.

First we shall write this using Java.

FileToRddExample.java

```
import org.apache.spark.SparkConf;
import org.apache.spark.api.java.JavaRDD;
import org.apache.spark.api.java.JavaSparkContext;


public class FileToRddExample {


   public static void main(String[] args) {
      // configure spark
      SparkConf sparkConf = new SparkConf().setAppName("Read Multiple Text Files to RDD")
                              .setMaster("local[2]").set("spark.executor.memory","2g");
      // start a spark context
      JavaSparkContext sc = new JavaSparkContext(sparkConf);


      // provide text file paths to be read to RDD, separated by comma
      String files = "data/rdd/input/file1.txt, data/rdd/input/file2.txt, data/rdd/input/file3.txt";


      // read text files to RDD
      JavaRDD<String> lines = sc.textFile(files);
```

```
        // collect RDD for printing
        for(String line:lines.collect()){
            System.out.println(line);
        }
    }
}
```

O/p

```
18/02/10 12:13:26 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks have all completed, from pool
18/02/10 12:13:26 INFO DAGScheduler: ResultStage 0 (collect at FileToRddExample.java:21) finished in 0.613 s
18/02/10 12:13:26 INFO DAGScheduler: Job 0 finished: collect at FileToRddExample.java:21, took 0.888843 s
This is File 1
Welcome to TutorialKart
Learn Apache Spark
Learn to work with RDD
This is File 2
Learn to read multiple text files to a single RDD
This is File 3
Learn to read multiple text files to a single RDD
18/02/10 12:13:26 INFO SparkContext: Invoking stop() from shutdown hook
18/02/10 12:13:26 INFO SparkUI: Stopped Spark web UI at http://192.168.1.104:4040
18/02/10 12:13:26 INFO MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
```

Conclusion:
In this way students can develop an application to Read multiple text files into single RDD using Spark