MANIPAL INSTITUTE OF TECHNOLOGY
Manipal – 576 104

DEPARTMENT OF INFORMATION & COMMUNICATION TECHNOLOGY

Data Analytics Lab
[ICT 4111]
For
Seventh Semester B. Tech. (IT) Degree

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

CONTENTS

Course Objectives

- To perform data analysis with R
- To get familiar with Hadoop eco system
- To understand the use of various tools on Hadoop platform
- To analyze unstructured and structured data

Course Outcomes

At the end of this course, students will be able to

- Perform exploratory and predictive analytics using R
- To write basic map reduce programs
- Using suitable tool for a given data
- To analyze the given data using Hadoop Ecosystem
- Analyze data using Big data tools

Evaluation Plan

- Internal Assessment Marks : 60%
  - ✓ Continuous evaluation for lab exercises (biweekly):10 marks (total 40 marks)
  - ✓ Evaluation of artifacts of the project and viva for the project (20 marks)
  - ✓ The assessment will depend on punctuality, program execution, maintaining the observation note, answering the questions in viva voce and submission of artifacts for the project.
- End semester assessment of 2 hour duration and project demonstration: 40%

INSTRUCTIONS TO THE STUDENTS

Pre- Lab Session Instructions

1. Students should carry the lab manual and the required stationery to every lab session
2. Be in time and follow the institution dress code
3. Must sign in the log register provided
4. Make sure to occupy the allotted seat and answer the attendance
5. Adhere to the rules and maintain the decorum

In- Lab Session Instructions

- Follow the instructions on the allotted exercises
- Show the program and results to the instructors on completion of experiments
- On receiving approval from the instructor, copy the program and results in the lab manual
- Prescribed textbooks and reference materials can be kept ready if required.
- Implement the lab exercises using R and Hadoop ecosystem
- Carry out the project using R and Hadoop Ecosystem

General Instructions for the exercises in Lab

- Implement the given exercise individually and not in a group.
- The programs should meet the following criteria:
    - o Programs should be interactive with appropriate prompt messages, error messages if any, and descriptive messages for outputs.

  - Programs should perform input validation (data type, range error, etc.) and give appropriate error messages and suggest corrective actions.
  - Comments should be used to give the statement of the problem and every function should indicate the purpose of the function, inputs and outputs.
  - Statements within the program should be properly indented.
  - Use meaningful names for variables and functions.
  - Make use of constants and type definitions wherever needed.
- Plagiarism (copying from others) is strictly prohibited and would invite severe penalty in evaluation.
- The exercises for each week are divided under three sets:
  - Solved exercise
  - Lab exercises : to be completed during lab hours
  - Additional Exercises - to be completed outside the lab or in the lab to enhance the skill
- If a student misses a lab class then he/she must ensure that the experiment is completed during the repetition class in case of genuine reason (medical certificate approved by HOD) with the permission of the faculty concerned
- Questions for lab tests and examination are not necessarily limited to the questions in the manual, but may involve some variations and / or combinations of the questions.
- A sample note preparation is given as a model for observation.

THE STUDENTS SHOULD NOT

- Bring mobile phones or any other electronic gadgets to the lab.
- Go out of the lab without permission.

LAB NO: 1 **Date:**

INTRODUCTION TO R, DATA GATHERING

Objectives:

1. To understand basic data analysis functions in R
2. Manipulate data within R

Introduction

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. Among other things it has

- an effective data handling and storage facility.
- a suite of operators for calculations on arrays, matrices.
- a large, coherent, integrated collection of intermediate tools for data analysis.
- graphical facilities for data analysis and display either directly at the computer or on hardcopy.
- a well developed, simple and effective programming language (called 'S') which includes conditionals, loops, user defined recursive functions and input and output facilities.

R is an interpreted language. All commands typed on the keyboard are directly executed without requiring to build a complete program. Syntax of R is very simple and intuitive.

When you use the R program it issues a prompt when it expects input commands. The default prompt is '>', which on UNIX might be the same as the shell prompt, and so it may appear that nothing is happening.

1.1 Setting up your working directory

```
getwd()
setwd(dir)
```

`getwd` returns a character string or `NULL` if the working directory is not available. On Windows the path returned will use `/` as the path separator and be encoded in UTF-8. The path will not have a trailing `/`unless it is the root directory (of a drive or share on Windows).

`setwd` returns the current directory before the change, invisibly and with the same conventions as `getwd`. It will give an error if it does not succeed (including if it is not implemented).

1.2 Getting help

The `help()` function and `?help` operator in R provide access to the documentation pages for R functions, data sets, and other objects, both for packages in the standard R distribution and for contributed packages.

To access documentation for the standard `lm` (linear model) function, for example, enter the command `help(lm)` or `help("lm")`, or `?lm` or `?"lm"` (i.e., the quotes are optional).

1.3 Installing packages

Many libraries are available in R which has to be installed explicitly depending on the requirement.

```
> install.packages(“package_name”)      #install package
> library(“package_name”)        #to use the installed package
```

1.4 R Commands and Case sensitivity

Technically R is an expression language with a very simple syntax. It is case sensitive as are most UNIX based packages, so A and a are different symbols and would refer to different variables. The set of symbols which can be used in R names depends on the operating system and country within which R is being run (technically on the locale in use). Normally all alphanumeric symbols are allowed plus ‘.’ and ‘_’, with the restriction that a name must start with ‘.’ or a letter, and if it starts with ‘.’ the second character must not be a digit. Names are effectively unlimited in length. Elementary commands consist of either expressions or assignments. If an expression is given as a command, it is evaluated, printed and the value is lost. An assignment also evaluates an expression and passes the value to a variable but the result is not automatically printed.
Commands are separated either by a semi-colon (‘;’), or by a newline. Elementary commands can be grouped together into one compound expression by braces (‘{’

and ‘}’). Comments can be put almost anywhere, starting with a hashmark (‘#’), everything to the end of the line is a comment. If a command is not complete at the end of a line, R will give a different prompt, by default + on second and subsequent lines and continue to read input until the command is syntactically complete. This prompt may be changed by the user. We will generally omit the continuation prompt and indicate continuation by simple indenting. Command lines entered at the console are limited to about 4095 bytes.

1.5 Objects

R works with objects which are, characterized by their names and their content, but also by attributes which specify the kind of data represented by an object. In order to understand the usefulness of these attributes, consider a variable that takes the value 1, 2, or 3: such a variable could be an integer variable (for instance, the number of students in a class), or the coding of a categorical variable (for instance, blood type of a person A, B, AB or O). It is clear that the statistical analysis of this variable will not be the same in both cases: with R, the attributes of the object give the necessary information.

All objects have two intrinsic attributes: mode and length. The mode is the basic type of the elements of the object; there are four main modes: numeric, character, complex, and logical (FALSE or TRUE). Other modes exist but they do not represent data, for instance function or expression. The length is the number of elements of the object. To display the mode and the length of an object, one can use the functions mode and length, respectively:

```
> x<-1
> mode(x)
[1] “numeric”
> length(x)
[1] 1
> str1 <- “Analytics”; bool1<- TRUE; com1 <- 1i
> mode(str1); mode(bool1); mode(com1)
[1] “character”
[1] “logical”
[1] “complex”
```

Whatever the mode, missing data are represented by NA (not available). A very large numeric value can be specified with an exponential notation:

```
> N <- 2.1e23
> N
[1] 2.1e+23
```

R correctly represents non-infinite numeric values, such as ∞ with Inf and -Inf, or values which are not numbers with NaN (not a number).

```
> x <- 5/0
> x
[1] Inf
> exp(x)
[1] Inf
> exp(-x)
[1] 0
> x - x
[1] NaN
```

A value of mode character is input with double quotes `"`. It is possible to include this latter character in the value if it follows a backslash. The two characters altogether `\n` will be treated in a specific way by some functions such as cat for display on screen, or write.table to write on the disk.

```
> x <- "Double quotes \" delimitate R's strings.\n"
> x
[1] "Double quotes \" delimitate R's strings.\n"
> cat(x)
Double quotes " delimitate R's strings.
>
```

Double quotes " delimitate R's strings. Alternatively, variables of mode character can be delimited with single quotes ('); in this case it is not necessary to escape double quotes with backslashes (but single quotes must be!):

```
> x <- 'Double quotes " delimitate R\'s strings.'
> x
[1] "Double quotes \" delimitate R's strings."
> cat(x)
Double quotes " delimitate R's strings.>
```

In R, the variables are not declared as some data type. The variables are assigned with R objects and the data type of the R object becomes the data type of the variable. There are many types of R objects. The frequently used ones are − Vectors, Lists, Matrices, Arrays, Factors, Data Frames. The simplest of these objects is the vector object.

There are six data types of these atomic vectors, also termed as six classes of vectors. The other R objects are built upon the atomic vectors as shown in **Table 1 Atomic vectors.**

Table 1 Atomic vectors

Data Type	**Example**
Logical	TRUE, FALSE
Numeric	12.3, 5, 999
Integer	2L, 34L, 0L
Complex	3 + 2i
Character	'a' , '"good", "TRUE", '23.4'
Raw	"Hello" is stored as 48 65 6c 6c 6f

Example :

```
> bool<-TRUE          # <- is assignment operator.
> class(bool)         # class(var) gives type of var.
[1] "logical"         # output
> x<-1
> class(x)            # is same as print(class(x))
[1] "numeric"
> y<-1L
> class(y)
[1] "integer"
> is.numeric(y)        #is.atomicVector(var) give either TRUE or FALSE
                       as output depending on the type of var passed
[1] TRUE
> is.logical(y)
[1] FALSE
> is.integer(y)        #any number suffixed with L forces the value to
be stored as integer.
[1] TRUE
> str1 <- charToRaw("DataAnalytics")     # converts to Raw type
> class(str1)
```

```
[1] "raw"
> str1
 [1] 44 61 74 61 41 6e 61 6c 79 74 69 63 73
```

"integer" is a subset of "numeric". Integers only go to a little more than 2 billion, while the other numerics can be much bigger. They can be bigger because they are stored as double precision floating point numbers.

The following table **Table 2 Type of objects** gives an overview of the type of objects representing data.

Table 2 Type of objects

Object	modes	Several modes possible in the same object?	Brief
vector	Numeric, character, complex, or logical	No	The function vector, which has two arguments mode and length. vector() creates a vector which elements have a value depending on the mode. Mode is specified as argument: 0 if numeric, FALSE if logical, or "" if character. The following functions have exactly the same effect and have for single argument the length of the vector: numeric(), logical(), and character().
factor	Numeric or character	No	Factors are used to create categorical data.
array	Numeric, character, complex or logical	No	arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension.

data frame	Numeric, character, complex or logical	Yes	Data frames are tabular data objects. In data frame each column can contain different modes of data.
list	Numeric, character, complex, logical, function, expression …	Yes	A list is created in a way similar to data frames with the function list. There is no constraint on the objects that can be included.
ts	Numeric, character, complex or logical	No	Time series data.

A vector is a variable in the commonly admitted meaning. A factor is a categorical variable. An array is a table with k dimensions, a matrix being a particular case of array with k = 2. Note that the elements of an array or of a matrix are all of the same mode. A data frame is a table composed with one or several vectors and/or factors all of the same length but possibly of different modes. A `ts' is a time series data set and so contains additional attributes such as frequency and dates. Finally, a list can contain any type of object, included lists!
For a vector, its mode and length are sufficient to describe the data. For other objects, other information is necessary and it is given by non-intrinsic attributes. Among these attributes, one can cite dim which corresponds to the dimensions of an object. For example, a matrix with 2 lines and 2 columns has for dim the pair of values [2, 2], but its length is 4.

General form :

```
factor(x, levels = sort(unique(x), na.last = TRUE), labels = levels,
exclude = NA, ordered = is.ordered(x))
```

levels specifies the possible levels of the factor (by default the unique values of the vector x), labels defines the names of the levels, exclude the values of x to exclude from the levels, and ordered is a logical argument specifying whether the levels of the factor are ordered.

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
```

The option byrow indicates whether the values given by data must fill successively the columns (the default) or the rows (if TRUE). The option dimnames allows to give names to the rows and columns.

Example :

```
> #numeric vector
> a<-vector(mode="numeric", length=10)
> a<-1:10    #assign values
> a      #default action is print. No need to use print() explicitly
[1]  1  2  3  4  5  6  7  8  9 10
> mode(a)
[1] "numeric"
> x <- c(1, 5, 4, 9, 0)     #another method to create vector
> length(x)
[1] 5
> #character vector
> apple <- c('red','green',"yellow")
> print(apple)       #explicit usage of print()
[1] "red"    "green"  "yellow"
> class(apple)
[1] "character"
> typeof(apple)
[1] "character"


>#usage of factor
>  data  =  c("apple",  "orange",  "pineapple","orange","apple",
"pineapple", "apple")
> fdata = factor(data)
> fdata
[1] apple     orange    pineapple orange    apple     pineapple apple
Levels: apple orange pineapple
> data = c(1,2,2,3,1,2,3,3,1,2,3,3,1)
> fdata = factor(data)
> fdata
 [1] 1 2 2 3 1 2 3 3 1 2 3 3 1
Levels: 1 2 3
> rdata = factor(data,labels=c("I","II","III"))
> rdata
 [1] I   II  II  III I   II  III III I   II  III III I
Levels: I II III

>#Create an array.
> arr1 <- array(c(1,2,3,4,5), dim=5)    #dim(dimension) represents
the length of the array in 1-D.
> arr1
[1] 1 2 3 4 5
>a <- array(c('green','yellow'),dim = c(3,3,2)) #creates 3-D array
>a
, , 1

     [,1]     [,2]     [,3]
[1,] "green"  "yellow" "green"
[2,] "yellow" "green"  "yellow"
[3,] "green"  "yellow" "green"

, , 2

     [,1]     [,2]     [,3]
[1,] "yellow" "green"  "yellow"
[2,] "green"  "yellow" "green"
[3,] "yellow" "green"  "yellow"


> # Create the data frame.
> # Method 1
>BMI <- data.frame(
   gender = c("Male", "Male","Female"),
   height = c(152, 171.5, 165),
   weight = c(81,93, 78),
   Age = c(42,38,26)
)
>BMI
  gender height weight Age
1   Male  152.0     81  42
2   Male  171.5     93  38
3 Female  165.0     78  26
># Method 2
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc")
> b = c(TRUE, FALSE, TRUE)
> df = data.frame(n, s, b)
> df
```

```
  n  s     b
1 2 aa  TRUE
2 3 bb FALSE
3 5 cc  TRUE
> ncol(df)     # ncol() returns number of columns
[1] 3
> nrow(df)    # nrow() returns number of rows
[1] 3
> df[2,1]      #accessing particular value
[1] 3
> df[2,3]
[1] FALSE

> #creating list
> list1 <- list(c(2,5,3),21.3,rep)
> list1
[[1]]
[1] 2 5 3

[[2]]
[1] 21.3

[[3]]
function (x, ...)  .Primitive("rep")

>#Create time series data
> age <- c(11,2,34,54,23,12,34,65,45,76,76,45)
> agets <- ts(age)      #One can mention the frequency of the
                        ts data. For monthly time series data,
                        you set frequency=12
> agets
Time Series:
Start = 1
End = 12
Frequency = 1
 [1]  11  2 34 54 23 12 34 65 45 76 76 45
```

The main difference between class and typeof is that the first can be defined by the user, but the type cannot.

```
> x<-list("a",c(1,2))
> # x is a list
> class(x)
[1] "list"
```

```
> # class can be user defined
> class(x)<-"newclass"
> class(x)
[1] "newclass"
> typeof(x)
[1] "list"
> typeof(x)<-"newclass"
Error in typeof(x) <- "newclass":could not find function "typeof<-"
```

1.6 Reading & Saving Data

R can read data stored in text (ASCII) files with the following functions: read.table, scan and read.fwf. R can also read files in other formats (Excel, SAS, SPSS, . . .), and access SQLtype databases, but the functions needed for this are to be installed explicitly.

The function read.table has for effect to create a data frame, and so is the main way to read data in tabular form. For instance, if one has a file named data.txt, the command:

```
> mydata <- read.table("data.txt")
```

will create a data frame named mydata, and each variable will be named, by default, V1, V2, . . . and can be accessed individually by mydata$V1, mydata$V2, . . . , or by mydata["V1"], mydata["V2"], . . . , or, still another solution, by mydata[, 1], mydata[,2],. . .

```
read.table (file, header = FALSE, sep = "", quote = "\"'", dec =
".", row.names, col.names, as.is = FALSE, na.strings = "NA",
colClasses = NA, nrows = -1, skip = 0, check.names = TRUE, fill =
!blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE,
comment.char = "#")
```

file	the name of the file (within "" or a variable of mode character), possibly with its path (the symbol \ is not allowed and must be replaced by /, even under Windows), or a remote access to a file of type URL (http://...)

header	a logical (FALSE or TRUE) indicating if the file contains the names of the variables on its first line
sep	the field separator used in the file, for instance sep="nt" if it is a tabulation.
quote	the characters used to cite the variables of mode character
dec	the character used for the decimal point
row.names	a vector with the names of the lines which can be either a vector of mode character, or the number (or the name) of a variable of the file (by default: 1, 2, 3, . . .)
col.names	a vector with the names of the variables (by default: V1, V2, V3,. . .)
as.is	controls the conversion of character variables as factors (if FALSE) or keeps them as characters (TRUE); as.is can be a logical, numeric or character vector specifying the variables to be kept as character
na.strings	the value given to missing data (converted as NA)
colClasses	a vector of mode character giving the classes to attribute to the columns
nrows	the maximum number of lines to read (negative values are ignored)
skip	the number of lines to be skipped before reading the data
check.names	if TRUE, checks that the variable names are valid for R
fill	if TRUE and all lines do not have the same number of variables, “blanks" are added
strip.white	(conditional to sep) if TRUE, deletes extra spaces before and after the character variables
blank.lines.skip	if TRUE, ignores “blank" lines
comment.char	a character defining comments in the data file, the rest of the line after this character is ignored (to disable this argument, use comment.char = "")

The variants of read.table are useful since they have different default values:

```
    read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".",
             fill = TRUE, ...)
   read.csv2(file, header = TRUE, sep = ";", quote="\"", dec=",",
             fill = TRUE, ...)
   read.delim(file, header = TRUE, sep = "\t", quote="\"", dec=".",
              fill = TRUE, ...)
   read.delim2(file, header = TRUE, sep = "\t", quote="\"", dec=",",
    fill = TRUE,...)
```

To write in a simpler way an object in a file, the command `write(x, file="data.txt")` can be used, where x is the name of the object (which can be a vector, a matrix, or an array). There are two options: nc (or ncol) which defines the number of columns in the file (by default nc=1 if x is of mode character, nc=5 for the other modes), and append (a logical) to add the data without deleting those possibly already in the file (TRUE) or deleting them if the file already exists (FALSE, the default).

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",
eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE,
qmethod = c("escape", "double"))
```

x	the name of the object to be written
file	the name of the file (by default the object is displayed on the screen)
append	if TRUE adds the data without erasing those possibly existing in the file
quote	a logical or a numeric vector: if TRUE the variables of mode character and the factors are written within " ", otherwise the numeric vector indicates the numbers of the variables to write within " " (in both cases the names of the variables are written within "" but not if quote = FALSE)
sep	the field separator used in the file
eol	the character to be used at the end of each line ("\n" is a carriage-return)

na	the character to be used for missing data
dec	the character used for the decimal point
row.names	a logical indicating whether the names of the lines are written in the file
col.names	id. for the names of the columns
qmethod	specifies, if quote=TRUE, how double quotes " included in variables of mode character are treated: if "escape" (or "e", the default) each " is replaced by \", if "d" each " is replaced by ""

1.7 Generating data

Regular sequences

A regular sequence of integers, for example from 1 to 30, can be generated with:

```
> x <- 1:30
```

The resulting vector x has 30 elements. The operator ‘:’ has priority on the arithmetic operators within an expression:

```
> 1:10-1
[1] 0 1 2 3 4 5 6 7 8 9
> 1:(10-1)
[1] 1 2 3 4 5 6 7 8 9
```

The function seq can generate sequences of real numbers as follows:

```
> seq(1, 5, 0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

where the first number indicates the beginning of the sequence, the second one the end, and the third one the increment to be used to generate the sequence. One can use also:

```
> seq(length=9, from=1, to=5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

One can also type directly the values using the function c:

```
> c(1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
15
```

It is also possible, if one wants to enter some data on the keyboard, to use the function scan with simply the default options:

```
> z <- scan()
1: 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0   #enter data though keyboard
10:                                #press enter to exit
Read 9 items
> z
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

The function rep creates a vector with all its elements identical:

```
> rep(1, 30)
[1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

The function sequence creates a series of sequences of integers each ending by the numbers given as arguments:

```
> sequence(4:5)
[1] 1 2 3 4 1 2 3 4 5
> sequence(4:9)
 [1] 1 2 3 4 1 2 3 4 5 1 2 3 4 5 6 1 2 3 4 5 6 7 1 2 3 4 5 6 7 8 1
     2 3 4 5 6 7 8 9
> sequence(c(10,5))
[1] 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5
```

Random sequences

Gaussian (normal) `rnorm(n, mean=0, sd=1)`
exponential `rexp(n, rate=1)`
gamma `rgamma(n, shape, scale=1)`
Poisson `rpois(n, lambda)`

It is useful in statistics to be able to generate random data, and R can do it for a large number of probability density functions.

Most of these functions have counterparts obtained by replacing the letter r with d, p or q to get, respectively, the probability density (dfunc(x, ...)), the cumulative probability density (pfunc(x, ...)), and the value of quantile (qfunc(p, ...), with 0 < p

< 1). The last two series of functions can be used to find critical values or P-values of statistical tests.

1.8 Subsetting

R has powerful indexing features for accessing object elements. These features can be used to select and exclude variables and observations.

Selecting (Keeping) Variables

```
# select variables v1, v2, v3
myvars <- c("v1", "v2", "v3")
newdata <- mydata[myvars]

# another method
myvars <- paste("v", 1:3, sep="")
newdata <- mydata[myvars]

# select 1st and 5th thru 10th variables
newdata <- mydata[c(1,5:10)]
```

Excluding (DROPPING) variables

```
# exclude variables v1, v2, v3
myvars <- names(mydata) %in% c("v1", "v2", "v3")
newdata <- mydata[!myvars]

# exclude 3rd and 5th variable
newdata <- mydata[c(-3,-5)]

# delete variables v3 and v5
mydata$v3 <- mydata$v5 <- NULL
```

Selecting Observations

```
# first 5 observations
newdata <- mydata[1:5,]
# based on variable values
newdata <- mydata[ which(mydata$gender=='F' & mydata$age > 65), ]
# or
attach(mydata)
newdata <- mydata[ which(gender=='F' & age > 65),]
detach(mydata)
```

Selection using the Subset Function

The subset( ) function is the easiest way to select variables and observations. In the following example, we select all rows that have a value of age greater than or equal to 20 or age less then 10. We keep the ID and Weight columns.

```
# using subset function
newdata <- subset(mydata, age >= 20 | age < 10, select=c(ID,
Weight))
```

Random Samples

Use the sample( ) function to take a random sample of size n from a dataset.

```
# take a random sample of size 50 from a dataset mydata
# sample without replacement
mysample <- mydata[sample(1:nrow(mydata), 50,  replace=FALSE),]
```

1.9 Control Structures : If else, For loops, while, repeat, next and break.

The language has available a conditional construction of the form

```
> if (expr_1) expr_2 else expr_3
```

where expr_1 must evaluate to a single logical value and the result of the entire expression is then evident. The "short-circuit" operators && and || are often used as part of the condition in an if statement. Whereas & and | apply element-wise to vectors, && and || apply to vectors of length one, and only evaluate their second argument if necessary.

There is a vectorized version of the if/else construct, the ifelse function. This has the form ifelse(condition, a, b) and returns a vector of the same length as condition, with elements a[i] if condition[i] is true, otherwise b[i] (where a and b are recycled as necessary).

There is also a for loop construction which has the form

```
> for (name in expr_1) expr_2
```

where name is the loop variable. expr_1 is a vector expression, (often a sequence like 1:20), and expr_2 is often a grouped expression with its sub-expressions written in terms of the dummy name. expr_2 is repeatedly evaluated as name ranges through the values in the vector result of expr_1.

As an example, suppose ind is a vector of class indicators and we wish to produce separate plots of y versus x within classes. One possibility here is to use coplot(),1 which will produce an array of plots corresponding to each level of the factor. Another way to do this, now putting all plots on the one display, is as follows:

```
> xc <- split(x, ind)
> yc <- split(y, ind)
> for (i in 1:length(yc)) {
plot(xc[[i]], yc[[i]])              #graph
abline(lsfit(xc[[i]], yc[[i]]))     #line
}
```

(Note the function split() which produces a list of vectors obtained by splitting a larger vector according to the classes specified by a factor. This is a useful function, mostly used in connection with boxplots.)

Warning: for() loops are used in R code much less often than in compiled languages. Code that takes a 'whole object' view is likely to be both clearer and faster in R. Other looping facilities include the

```
> repeat expr
> while (condition) expr
```

The break statement can be used to terminate any loop, possibly abnormally. This is the only way to terminate repeat loops.

The next statement can be used to discontinue one particular cycle and skip to the "next".

1.10 Functions

As a first example, consider a function to calculate the two sample t-statistic, showing "all the steps". This is a sample example, since there are other, simpler ways of achieving the same end. The function is defined as follows:

```
> twosam <- function(y1, y2) {       #y1 and y2 are arguments to
                                      the function
n1 <- length(y1); n2 <- length(y2)
yb1 <- mean(y1); yb2 <- mean(y2)
s1 <- var(y1); s2 <- var(y2)
s <- ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2)
tst <- (yb1 - yb2)/sqrt(s*(1/n1 + 1/n2))
tst
}
```

With this function defined, you could perform two sample t-tests using a call such as

```
> tstat <- twosam(data$male, data$female); tstat
```

The functions available in R for manipulating data are too many to be listed here. One can find all the basic mathematical functions (log, exp, log10, log2, sin, cos, tan, asin, acos, atan, abs, sqrt, . . .), special functions (gamma, digamma, beta, besselI, . . .), as well as diverse functions useful in statistics. Some of these functions are listed in the following table **Table 3 Built in functions**.

Table 3 Built in functions

sum(x)	sum of the elements of x
prod(x)	product of the elements of x
max(x)	maximum of the elements of x
min(x)	minimum of the elements of x
which.max(x)	returns the index of the greatest element of x
which.min(x)	returns the index of the smallest element of x
range(x)	id. than c(min(x), max(x))
length(x)	number of elements in x
mean(x)	mean of the elements of x
median(x)	median of the elements of x
var(x) or cov(x)	variance of the elements of x, if x is a matrix or a data frame, the variance-covariance matrix is calculated
cor(x)	correlation matrix of x if it is a matrix or a data frame (1 if x is a vector)
var(x,y) or cov(x, y)	covariance between x and y, or between the columns of x and those of y if they are matrices or data

cor(x, y)	linear correlation between x and y, or correlation matrix if they are matrices or data frames

These functions return a single value (thus a vector of length one), except range which returns a vector of length two, and var, cov, and cor which may return a matrix. The following functions which are listed in **Table 4 Built in functions**. return more complex results.

Table 4 Built in functions.

round(x, n)	rounds the elements of x to n decimals
rev(x)	reverses the elements of x
sort(x)	sorts the elements of x in increasing order; to sort in decreasing order: rev(sort(x))
rank(x)	ranks of the elements of x
log(x, base)	computes the logarithm of x with base base
scale(x)	if x is a matrix, centers and reduces the data; to center only use the option center=FALSE, to reduce only scale=FALSE (by default center=TRUE, scale=TRUE)
match(x, y)	returns a vector of the same length than x with the elements of x which are in y (NA otherwise)
which(x == a)	returns a vector of the indices of x if the comparison operation is true (TRUE), in this example the values of i for which x[i] == a (the argument of this function must be a variable of mode logical)
choose(n, k)	computes the combinations of k events among n repetitions
na.omit(x)	suppresses the observations with missing data (NA) (suppresses the corresponding line if x is a matrix or a data frame)
unique(x)	if x is a vector or a data frame, returns a similar object but with the duplicate elements suppressed

subset(x, ...)	returns a selection of x with respect to criteria (..., typically comparisons: x$V1 < 10); if x is a data frame, the option select gives the variables to be kept (or dropped using a minus sign)
sample(x, size)	resample randomly and without replacement size elements in the vector x, the option replace = TRUE allows to resample with replacement

1.11 Loop functions – lapply, apply, sapply

Data is simulated with rnorm function. The first has mean 0, second mean of 2, third of mean of 5, and with 30 rows.

```
> m <- matrix(data=cbind(rnorm(30, 0), rnorm(30, 2), rnorm(30, 5)),
nrow=30, ncol=3)
```

Apply

If data frame is being used then data types must all be the same otherwise they will be subjected to type conversion. This may or may not be what user want, if the data frame has string/character data as well as numeric data, the numeric data will be converted to strings/characters and numerical operations will probably not give the expected output.

Usage of Apply function depends on the end user requirement. Apply function is used to traverse the function which is passed as second argument either row wise or column wise.

In the following example three numbers are expected at the end, that is the mean value for each column.

```
> apply(m, 2, mean)
#[1] -0.02664418  1.95812458  4.86857792
```

Roughly, 0,2, and are the mean of each column as expected. Passing a 1 in the second argument, we get 30 values back, giving the mean of each row.

```
> apply(m, 1, mean)
 [1] 2.291269 2.165206 1.366036 2.157848 1.883528 2.637516 2.454606
```

```
     2.505342
 [9] 2.081313 2.537847 2.022317 2.632561 2.687647 2.382216 1.864432
     2.693109
[17] 2.714948 2.881620 2.404267 3.442941 2.590379 2.963917 1.978979
     2.326533
[25] 1.002473 2.479101 2.466446 2.676571 1.678714 3.249926
```

Following example gives total number of negative numbers in each column.

```
> apply(m, 2, function(x) length(x[x<0]))
[1] 13  0  0
```

So 13 negative values in column one, and none in column two and three.

Note that return value of function is specified. R will magically return the last evaluated value. The actual function is using subsetting to extract all the elements in x that are less than 0, and then counting how many are left are using length.

Let’s look at the mean value of only the positive values:

```
> apply(m, 2, function(x) mean(x[x>0]))
#[1] 0.7492143 2.1104836 4.9510160
```

Using sapply and lapply

These two functions work in a similar way, traversing over a set of data like a list or vector, and calling the specified function for each item.
When the traversal of function over the given data set is not required in linear fashion, sapply and lapply functions are useful.

Example

```
> sapply(1:3, function(x) x^2)
#[1] 1 4 9
```

lapply is very similar, however it will return a list rather than a vector:

```
lapply(1:3, function(x) x^2)
#[[1]]
#[1] 1
#
#[[2]]
```

```
#[1] 4
#
#[[3]]
#[1] 9
```

Passing simplify=FALSE to sapply will also give you a list:

```
sapply(1:3, function(x) x^2, simplify=F)
#[[1]]
#[1] 1
#
#[[2]]
#[1] 4
#
#[[3]]
#[1] 9
```

And one can use unlist with lapply to get a vector.

```
unlist(lapply(1:3, function(x) x^2))
#[1] 1 4 9
```

Mean of a data frame either row wise or column wise can be computed using sapply as shown in the following example :

```
> sapply(1:3, function(x) mean(m[,x]))   #column wise
[1] 0.06046096 2.11048364 4.95101599

> sapply(1:30, function(x) mean(m[x,]))   #row wise
 [1] 2.291269 2.165206 1.366036 2.157848 1.883528 2.637516 2.454606
     2.505342
 [9] 2.081313 2.537847 2.022317 2.632561 2.687647 2.382216 1.864432
     2.693109
[17] 2.714948 2.881620 2.404267 3.442941 2.590379 2.963917 1.978979
     2.326533
[25] 1.002473 2.479101 2.466446 2.676571 1.678714 3.249926

#another method
> sapply(1:3, function(x, y) mean(y[,x]), y=m)
[1] 0.06046096 2.11048364 4.95101599
```

References

1. “R for Beginners” --Emmanuel Paradis
2. An Introduction to R -- W. N. Venables, D. M. Smith and the R Core Team

3. https://stat.ethz.ch/R-manual
4. https://www.r-project.org/
5. https://www.tutorialspoint.com/r/r_data_types.htm

LAB NO: 2 **Date:**

PREDICTIVE ANALYSIS AND VISUALIZATION: R

Objectives:

1. To build analysis model for any given dataset using R.
2. To visualize the processed data using R.

2.1 Visualization

Many inbuilt functions and libraries are available to build visualization in R. Depending on the analytics requirement comparison (bar, column, line chart etc), composition (stacked, pie, waterfall chart etc), distribution(histogram, scatter etc) or relationship (scatter, bubble chart etc) presentation styles could be used.

Basic Plot

```
plot(x, y, …)
```

x	the coordinates of points in the plot.
y	the y coordinates of points in the plot, *optional* if x is an appropriate structure.
...	Arguments to be passed to methods, such as graphical paramenters. Many methods will accept the following arguments: **type** what type of plot should be drawn. Possible types are "p" for points, "l" for lines, "b" for both, "c" for the lines part alone of "b", "o" for both 'overplotted', "h" for 'histogram' like vertical lines, "s" for stair steps, "S" for other steps, "n" for no plotting. **main** an overall title for the plot **sub** a sub title for the plot **xlab** a title for the x axis **ylab** a title for the y axis **asp** the y/x aspect ratio.

Legend

This function can be used to add legends to plots.

```
     legend(x, y)                    #x, y coordinates on graph.
```

Example :

```
 > height <- c(145, 167, 176, 123, 150)
 > weight <- c(51, 63, 64, 40, 55)
 > plot(height,weight)                  #plot values read from vector
 > bmi<- cbind(height, weight)   #columnwise binding
 > write.csv(bmi, file="BMI.csv", row.names=FALSE)
 > df<-read.csv("BMI.csv")
 > plot(df$height, df$weight)   #plot values read from file
 > plot(df$height, df$weight, xlab="height", ylab="weight")
 > #Simple quantile
 > plot(x <- sort(rnorm(47)), type = "s", main = "plot(x, type =
            s)")
 > points(x, cex = .5, col = "dark red")
 > plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species,
   pch = 16, cex = 2)   #col specifies color, one can create a
                        vector of color and pass as argument value
 > legend(x = 4.5, y = 7, legend = levels(iris$Species), col =
   c(1:3), pch = 16)
```

Histogram

Histogram is used to plot continuous variable. It breaks the data into bins and shows frequency distribution of these bins.

```
 > hist(bmi, breaks=5, main="Breaks=5")
 > hist(bmi, main="Without Breaks")
```

Box Plot

Iris dataset is built in data set. To view onl 1st few lines one can make use of command head.

```
 > head(iris)   #shows 1st 5 rows
 > boxplot(iris$Sepal.Length ~ iris$Species)   # x ~ y
 > box()                       #puts outside box
```

Bar Plot

```
> barplot(iris$Sepal.Length)
```

Saving plot

```
> png("Sepal vs Petal Length in Iris.png", width = 500, height = 500,
res = 72)                #saves in png format
> plot(iris$Sepal.Length, iris$Petal.Length, col = iris$Species, main
       = "Sepal vs Petal Length in Iris")
> dev.off()
```

ggplot2

In ggplot, one can independently specify plot building blocks and combine them to create any kind of graphical display. Building blocks of a graph include: data, aesthetic mapping, geometric object, statistical transformations, scales, coordinate system, position adjustments and faceting. Aesthitic mapping includes position, color, fill (color), shape, linetype and size.

This function is available in package ggplot2. Also to search for available building blocks one can use

```
help.search("geom_", package = "ggplot2")
```

Example

```
> #basic plot iris dataset
> plot(Sepal.Length ~ Sepal.Width, data = subset(iris, Species ==
  "setosa"))
> points(Sepal.Length ~ Sepal.Width, data = subset(iris, Species ==
  "versicolor"), col="red")
> legend(4,5.8, c("setosa", "versicolor"), title="IRIS",
  col=c("black", "red"),pch=c(1,1))
> #using ggplot2
> p1<-ggplot(subset(iris, Species %in% c("setosa", "versicolor")),
  aes(x=Sepal.Length, y=Sepal.Width, color=Species)) + geom_point()
> p1 + geom_text(aes(label=Species), size=3)
> #line graph using ggplot
> ggplot(iris, aes(x = Petal.Width, y = Petal.Length, color=Species))
  geom_line()

> #histogram using hist()
> hist(iris$Petal.Width)
> #histogram usingn ggplot()
> ggplot(iris, aes(x = Petal.Width)) +  geom_histogram()
```

3.2 Predictive Analytics

Predictive analytics is an area of data mining that deals with extracting information from data and using it to predict trends and behavior patterns. Predictive analytics sometimes used synonymously with predictive modeling encompasses a variety of statistical techniques from modeling, machine learning, and data mining that analyze current and historical facts to make predictions about future, or otherwise unknown, events.

For all the algorithms function predict is used. Depending on the algorithm used a model has to fit the values on new variables which can be implemented using predict().

```
predict(object, newdata, se.fit = FALSE, scale = NULL, df = Inf,
     interval = c("none", "confidence", "prediction"), level =
     0.95, type = c("response", "terms"),terms = NULL, na.action =
     na.pass, pred.var = res.var/weights, weights = 1, ...)
```

object	Object of class inheriting from "lm"
newdata	An optional data frame in which to look for variables with which to predict. If omitted, the fitted values are used.
se.fit	A switch indicating if standard errors are required.
scale	Scale parameter for std.err. calculation.
df	Degrees of freedom for scale.
interval	Type of interval calculation. Can be abbreviated.

level	Tolerance/confidence level.
type	Type of prediction (response or model term). Can be abbreviated.
terms	If type = "terms", which terms (default is all terms), a character vector.
na.action	function determining what should be done with missing values in newdata. The default is to predict NA.
pred.var	the variance(s) for future observations to be assumed for prediction intervals. See 'Details'.
weights	variance weights for prediction. This can be a numeric vector or a one-sided model formula. In the latter case, it is interpreted as an expression evaluated in newdata.
...	further arguments passed to or from other methods.

3.2.1 Naïve Bayes classifier

The naive Bayes classifier greatly simplify learning by assuming that features are independent given class. It is used to compute the conditional a-posterior probabilities of a categorical class variable given independent predictor variables using the Bayes rule.

```
naiveBayes(formula, data, ..., subset, na.action = na.pass)
naiveBayes(x, y, ...)
```

The function naiveBayes is available in package e1071.

Example,

```
> #categorical data
> datavoters <- read.csv("voters.txt", head=FALSE)
> model <- naiveBayes(datavoters $V1 ~ ., data = datavoters)
> predict(model, datavoters [1:10,-1]) # -1 refers exclude 1st
                                          column
> predict(model, datavoters [1:10,], type="raw")
> pred <- predict(model, datavoters [,-1])
> table(pred, datavoters$V1)  #confusion martix
> ## Example with metric predictors:
> data(iris)
> m <- naiveBayes(Species ~ ., data = iris)  # . represents all
> ## alternatively:
> m <- naiveBayes(iris[,-5], iris[,5])  # -5 exclude 5th column
> m
> table(predict(m, iris[,-5]), iris[,5])  #confusion matrix
```

3.2.2 k-Means

k-means creates *k* groups from a set of objects so that the members of a group are more similar. It's a popular cluster analysis technique for exploring a dataset.

```
kmeans(x, centers, iter.max = 10, nstart = 1, algorithm =
c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"), trace=FALSE)
```

X	numeric matrix of data, or an object that can be coerced to such a matrix (such as a numeric vector or a data frame with all numeric columns).
centers	either the number of clusters, say k, or a set of initial (distinct) cluster centres. If a number, a random set of (distinct) rows in x is chosen as the initial centres.
iter.max	the maximum number of iterations allowed.
nstart	if centers is a number, how many random sets should be chosen?

Example:

```
> x <- data.frame(y1=rnorm(100, sd = 0.3),y2=rnorm(100, sd = 0.4))
 > ch <- kmeans(x, 2)      # 2 clusters
 > plot(x, col = ch$cluster)
 > points(ch$centers, col = 1:2, pch = 8, cex = 2)
 > #using ggplot
 > ggplot(x, aes(y1,y2, color = ch$cluster)) + geom_point()
 > #another example
 > ch <- kmeans(x, 5, nstart = 25)
 > plot(x, col = ch$cluster)
 > points(ch$centers, col = 1:5, pch = 8)
```

3.2.3 kNN

k-nearest neighbors is non parametric used for classification and regression.

```
knn(train, test, cl, k = 1, l = 0, prob = FALSE, use.all = TRUE)
```

train	matrix or data frame of training set cases.
test	matrix or data frame of test set cases. A vector will be interpreted as a row vector for a single case.
cl	factor of true classifications of training set
k	number of neighbours considered.
L	minimum vote for definite decision, otherwise doubt.
prob	If this is true, the proportion of the votes for the winning class are returned as attribute prob.
use.all	controls handling of ties. If true, all distances equal to the kth largest are included. If false, a random selection of distances equal to the kth is chosen to use exactly k neighbours.

This function is available in package class.

Example

```
> x1 <- c(7,7,3,1)
> x2 <- c(7,4,4,4)
> y <- c(1,1,2,2) #label
> df<- cbind(x1,x2)  #train dataset
> testDF <- c(3,7)  #test dataset
> pred <- knn(train = df, test = testDF, cl = y, k=3)
> pred  #shows the label of test data
```

3.2.4 Linear Regression

Linear regression is an approach for modelling the relationship between a scalar dependent variable and one or more explanatory variables.

```
 lm(formula, data)

  formula is a symbol presenting the relation between x and y
  data is the vector on which the formula will be applied.
```

Example

```
> library("MASS")   #data set cats is available in library MASS.
> data(cats)   #loads specified data set
> head(cats)
> summary(cats)
> plot(Hwt ~ Bwt, data=cats)
> cat.mod <- lm(Bwt ~ Hwt, data=cats)   #label is Bwt
> summary(cat.mod)
> testData <- data.frame(Hwt=3.5)
> predict(cat.mod, testData)
```

If model has to be built against many variables formula will be y ~ x1 + x2 + .. + xn. In the above example model is built by considering only one variable.

3.2.5 Decision Trees

Decision tree is a supervised learning algorithm mostly used in classification problems. It works for both categorical and continuous input and output variables. Split on each node depends on use of multiple algorithm like chi square, gini index, information gain and reduction in variance. Selection of these algorithm depends on target variable.

This package is available in rpart.

```
> df<-read.csv("adult.csv")
> smp_size <- floor(0.7 * nrow(df))
> set.seed(123)
> train_ind <- sample(seq_len(nrow(df)), size = smp_size)
> trainData <- df[train_ind, ]
> testData <- df[-train_ind, ]
> fit <- rpart(trainData$V15 ~ . , data = trainData, method =
"class")  #method value might differ depending the data type
> summary(fit)
> predicted=predict(fit, testData)
> actualTrend<-testData$V15
> model_pred_trend = rep(0,nrow(testData))
> for(i in 1:nrow(predicted))
  predictedPattern[i] <- if(predicted[i,1]<0.5) "<=50K" else ">50K"
> table(actualTrend,predictedPattern)
> #consider few observation and draw line graph
```

3.2.6 Logistic Regression

Logistic regression is a regression model where the dependent variable is categorical.

```
glm( formula, family=binomial)
```

Example :

```
> mydata<-read.csv("cancer.csv")
> mydata$wt.loss[is.na(mydata$wt.loss)] <-
mean(mydata$wt.loss,na.rm=TRUE) #replace na with mean of that
column
> mydata$meal.cal[is.na(mydata$meal.cal)]<-
mean(mydata$meal.cal,na.rm=TRUE)
> mydata$ph.ecog[is.na(mydata$ph.ecog)]<-
mean(mydata$ph.ecog,na.rm=TRUE)
> mydata$pat.karno[is.na(mydata$pat.karno)]<-
mean(mydata$pat.karno,na.rm=TRUE)
> mydata$ph.karno[is.na(mydata$ph.karno)]<-
mean(mydata$ph.karno,na.rm=TRUE)
> index <- 1:nrow(mydata)
> trainindex <- sample(index, trunc(length(index)/4))  #select only
1/4th population
> test <- mydata[-trainindex, ]  #exclude data with trainindex
> mylogit <- glm(status ~
inst+time+age+sex+ph.ecog+ph.karno+pat.karno+meal.cal+wt.loss,
family = binomial("logit"), data=train ) #loggistic regression
> summary.glm(mylogit)
> new<-glm(status ~ inst+sex+ph.ecog, family = binomial("logit"),
data=train )
```

```
> predt <- predict(new, test ,type = 'response')
> predt
```

Replace predt values with either 0 or 1 if original value is less than 0.5. Check the accuracy. Consider few observation and visualize.

References

1. http://rstudio-pubs-static.s3.amazonaws.com/7953_4e3efd5b9415444ca065b1167862c349.html
2. https://www.r-bloggers.com/7-visualizations-you-should-learn-in-r/
4. https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/
5. http://ugrad.stat.ubc.ca/R/library/e1071/html/naiveBayes.html
6. http://tutorials.iq.harvard.edu/R/Rgraphics/Rgraphics.html
7. https://www.analyticsvidhya.com/blog/2015/11/beginners-guide-on-logistic-regression-in-r/
8. Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science.

Dataset

https://drive.google.com/open?id=0B4gbuqYuod28WkJUaktGclVCZzA

LAB NO: 3 **Date:**

DATA PROCESSING AND ANALYSIS

Objectives:

1. Compare the predictive analysis algorithms
2. Choose suitable R function to perform any task.

1. Considering the dataset Online Retail, write R code for the following

   i. Read the table into a dataframe.
   ii. Find out the total number of items purchased by UK customers
   iii. What is the amount spent by customer '13744' on 20-02-2011.
   iv. Identify the item which has maximum sale in France.
   v. List top 5 items which had maximum sale in second quarter of year 2011.
   vi. What the average amount generated by retail shop during the month of April 2011. (HINT : amount generated per transaction = quantity * unit price)
   vii. Compare the total amount earned in third quarter of year 2010 and 2011.
   viii. How many customers have done online shopping in the month of December 2010.
   ix. Display the graph showing quarter wise performance achieved on the online sales. (HINT: performance is directly proportional to amount generated during the transaction.)

2. Considering Wine_data, write R code for the following

   i. Find which attribute has more correlation with quality of wine.
   ii. Find total number of observations made for the best quality wine. (highest number indicates the best quality)
   iii. Round off the residual sugar values. Identify how many unique factors are available in residual sugar.
   iv. Find the mean value of pH (use apply)

3. Considering Titanic dataset, write R code for the following

i. What is the average age of passengers who travelled in 3rd class chamber? What is the estimated variance of the ages?
ii. Find out the total number of male passengers who travelled in 1st class chamber whose age is unknown.
iii. How many female passengers have survived who were travelling in 2nd class chamber, whose age is between 10 to 20. Also display the name of the passengers.

4. Consider the spam dataset, apply classification algorithm and compare the accuracy. Assume 80% of the dataset as training dataset and remaining as testing dataset.

5. Consider television selling dataset, apply multiple linear regression. Visualize the analysis.

6. Consider 3D_spatial_network dataset. Apply clustering algorithm for the same. Visualize the analysis.

References

1. https://vincentarelbundock.github.io/Rdatasets/datasets.html
2. Daqing Chen, Sai Liang Sain, and Kun Guo, Data mining for the online retail industry: A case study of RFM model-based customer segmentation using data mining, Journal of Database Marketing and Customer Strategy Management, Vol. 19, No. 3, pp. 197â€“208, 2012 (Published online before print: 27 August 2012. doi: 10.1057/dbm.2012.17).
3. http://www.stat.ufl.edu/~winner/datasets.html
4. Building Accurate 3D Spatial Networks to Enable Next Generation Intelligent Transportation Systems Proceedings of International Conference on Mobile Data Management (IEEE MDM), June 3-6 2013, Milan, Italy

Datasets

https://drive.google.com/open?id=0B4gbuqYuod28X0k4NDNQbmpGUTA

LAB NO: 4 **Date:**

MAP REDUCE PROGRAMMING

Objectives:

1. To understand the basics of hadoop.
2. To learn basics of Map reduce programming.

1. Introduction to Hadoop

Apache Hadoop is an open source software platform for distributed storage and distributed processing of very large data sets on computer clusters built from commodity hardware. Hadoop services provide for data storage, data processing, data access, data governance, security, and operations.

Benefits

Some of the reasons organizations use Hadoop is its' ability to store, manage and analyze vast amounts of structured and unstructured data quickly, reliably, flexibly and at low-cost.

- **Scalability and Performance** – distributed processing of data local to each node in a cluster enables Hadoop to store, manage, process and analyze data at petabyte scale.
- **Reliability** – large computing clusters are prone to failure of individual nodes in the cluster. Hadoop is fundamentally resilient – when a node fails processing is re-directed to the remaining nodes in the cluster and data is automatically re-replicated in preparation for future node failures.
- **Flexibility** – unlike traditional relational database management systems, you don't have to created structured schemas before storing data. You can store data in any format, including semi-structured or unstructured formats, and then parse and apply schema to the data when read.
- **Low Cost** – unlike proprietary software, Hadoop is open source and runs on low-cost commodity hardware.

High level architecture of the Hadoop as shown in the Figure 1 High level architecture of Hadoop . There are two primary components at the core of Apache Hadoop 1.x: the

Hadoop Distributed File System (HDFS) and the MapReduce parallel processing framework.

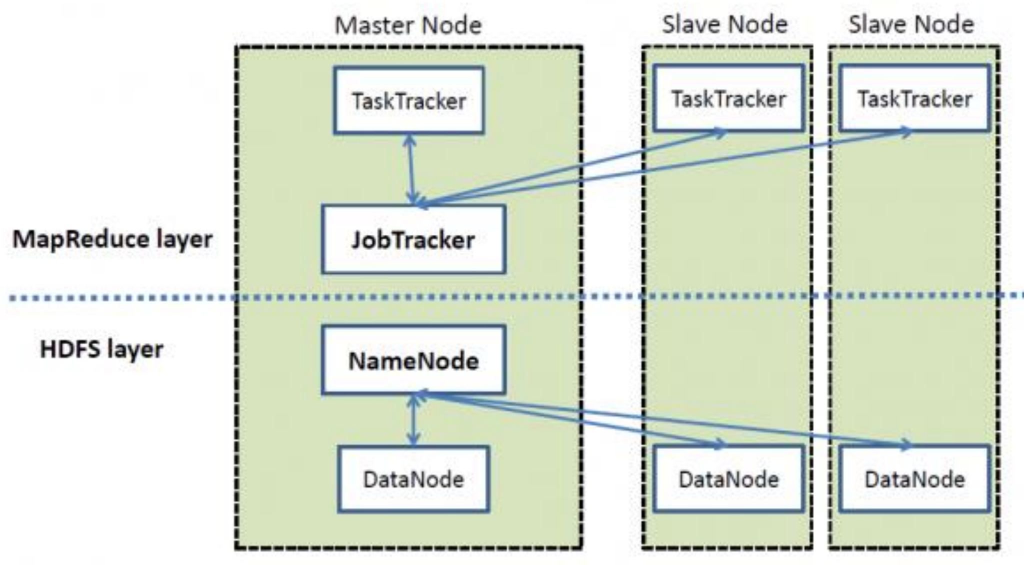


Figure 1 High level architecture of Hadoop

Login to hduser : su –hduser
Start cluster: start-all.sh

HDFS Commands

1. **Create a directory in HDFS at given path**
   hadoop fs -mkdir <paths>

2. **List the contents of a directory.**
   hadoop fs -ls <args>

3. **Copy a file from/To Local file system to HDFS**
   hadoop fs -copyFromLocal <localsrc> URI
   hadoop fs -copyToLocal [-ignorecrc] [-crc] URI <localdst>

4. **Move file from source to destination**
   hadoop fs -mv <src> <dest>

5. **Remove a file or directory in HDFS.**

Remove files specified as argument. Deletes directory only when it is empty
hadoop fs -rm <arg>

Recursive version of delete.
hadoop fs -rmr <arg>

6. **Display last few lines of a file.**
   hadoop fs -tail <path[filename]>

7. **Print the contents of the file on the terminal**
   hadoop fs -cat <path[filename]>

2. Introduction to Map Reduce

MapReduce is a framework designed for writing programs that process large volume of structured and unstructured data in parallel fashion across a cluster, in a reliable and fault-tolerant manner. MapReduce concept is simple to understand who are familiar with distributed processing framework.

MapReduce works with Key-Value pair. In the context of Hadoop, keys are associated with values. This data in MapReduce is stored in such a way that the values can be sorted and rearranged (Shuffle and sort wrt to MapReduce) across a set of keys. All data emitted in the flow of a MapReduce program is in the form of pairs. Some important features of key/value data will become apparent are:

1. Keys must be unique.
2. Each value must be associated with a key
3. A key can have no values also.

Key-value data is the foundation of MapReduce paradigm which means much of the data is in key-value nature or we can represent it in such a way. In short we can say that data model to be applied for designing MapReduce program is Key-Value pair.
It uses Divide and Conquer technique to process large amount of data. It divides input task into smaller and manageable sub-tasks (They should be executable independently) to execute them in-parallel.
MapReduce Algorithm uses the following three main steps:

1. Map Function
2. Shuffle Function
3. Reduce Function

Map Function is the first step in MapReduce Algorithm. It takes input tasks and divides them into smaller sub-tasks. Then perform required computation on each sub-task in parallel.

This step performs the following two sub-steps:

1. Splitting
2. Mapping

- Splitting step takes input Data Set from Source and divide into smaller Sub-Data Sets.
- Mapping step takes those smaller Sub-Data Sets and perform required action or computation on each Sub-Data Set.

The output of this Map Function is a set of key and value pairs as <Key, Value> as shown in the Figure 2 Key value pair generation.

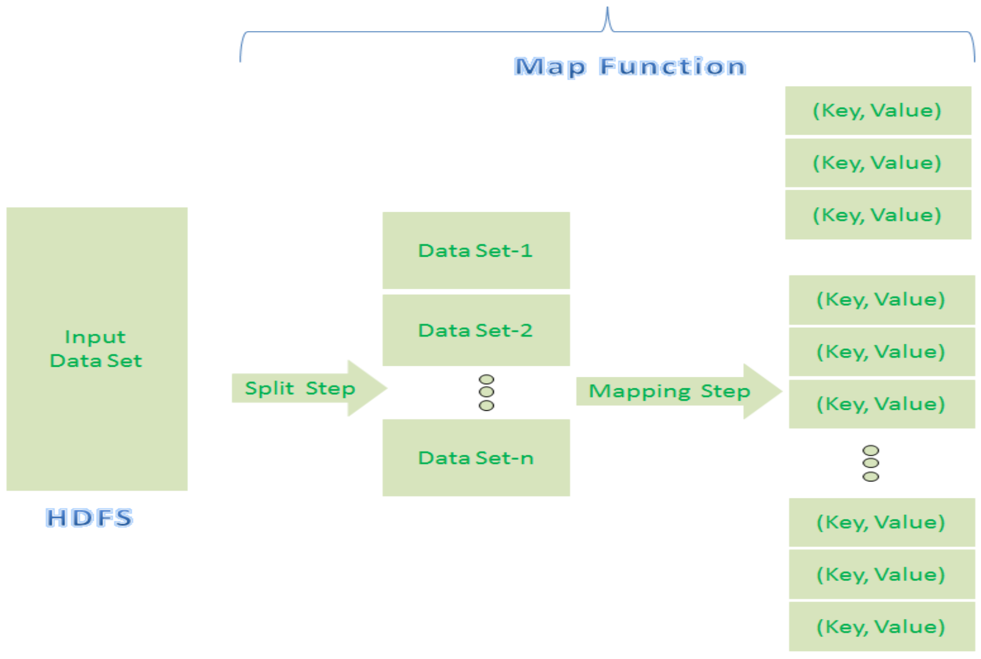


Figure 2 Key value pair generation

Shuffle Function

It is the second step in MapReduce Algorithm. Shuffle Function is also know as "Combine Function".

It performs the following two sub-steps:

1. Merging

2. Sorting

It takes a list of outputs coming from "Map Function" and perform these two sub-steps on each and every key-value pair.

- Merging step combines all key-value pairs which have same keys (that is grouping key-value pairs by comparing "Key"). This step returns <Key, List<Value>>.
- Sorting step takes input from Merging step and sort all key-value pairs by using Keys. This step also returns <Key, List<Value>> output but with sorted key-value pairs.

Finally, Shuffle Function returns a list of <Key, List<Value>> sorted pairs to next step. Generation key, list<value> is shown in Figure 3 Shuffle function.

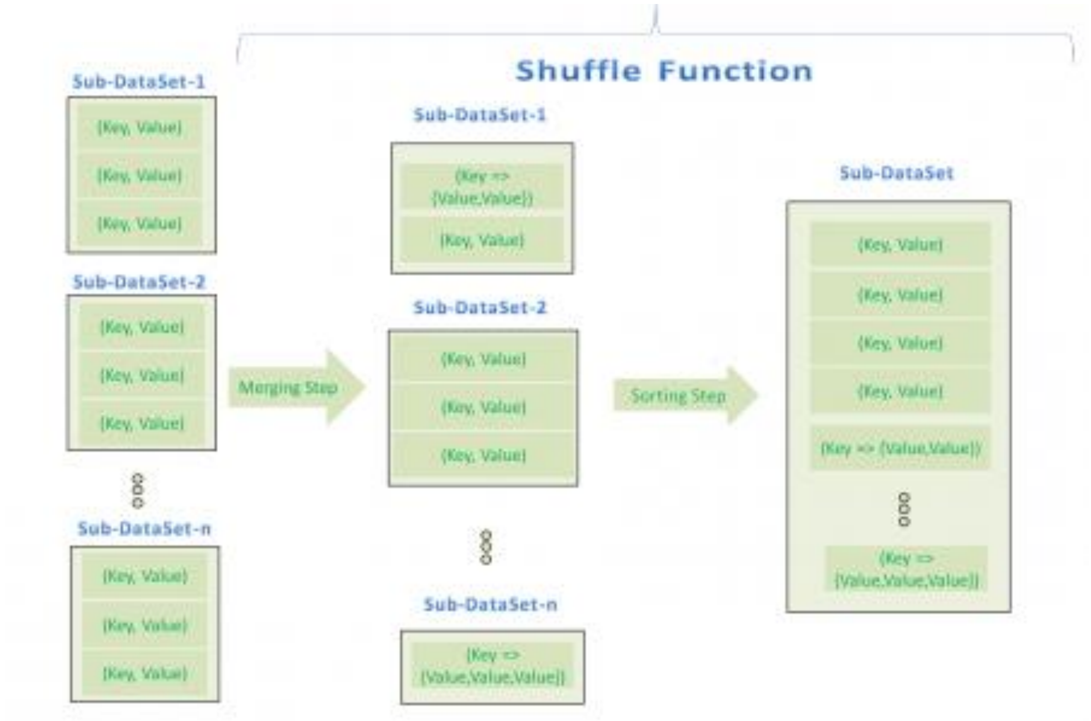


Figure 3 Shuffle function

MapReduce Second Step Output:

Shuffle Function Output = List of <Key, List<Value>> Pairs

Reduce Function

It is the final step in MapReduce Algorithm. It performs only one step : Reduce step. It takes list of <Key, List<Value>> sorted pairs from Shuffle Function and perform reduce operation as shown in Figure 4 Reduce function

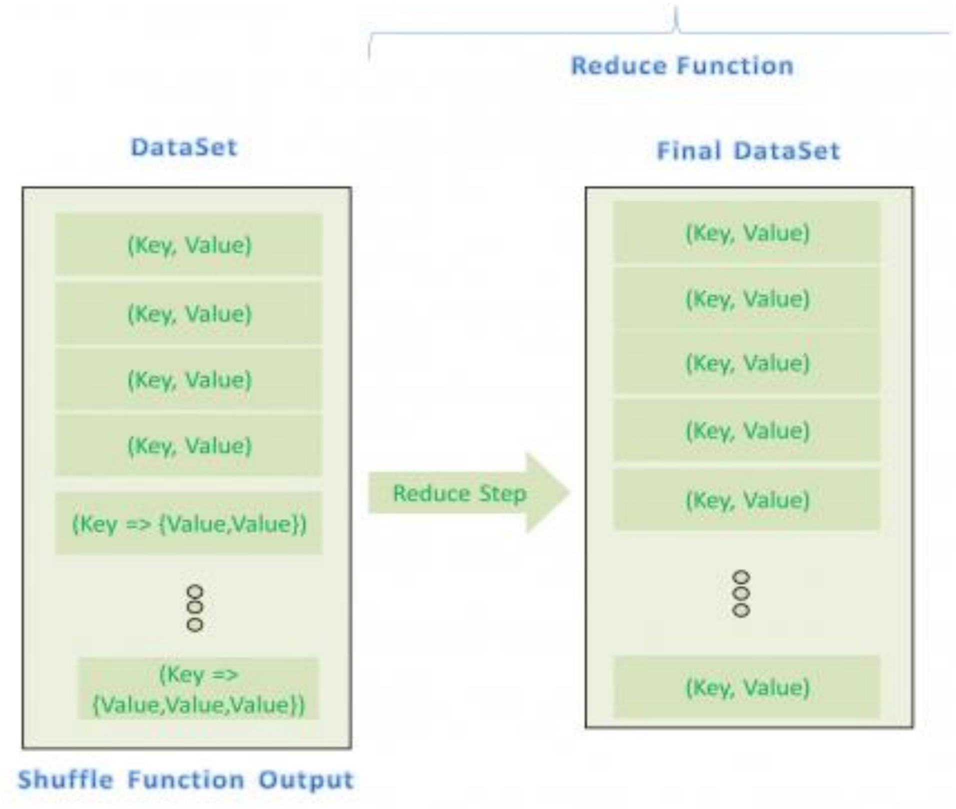


Figure 4 Reduce function

MapReduce Final Step Output:

Reduce Function Output = List of <Key, Value> Pairs

Final step output looks like first step output. However final step <Key, Value> pairs are different than first step <Key, Value> pairs. Final step <Key, Value> pairs are computed and sorted pairs.

Solved Exercise:

- Count the number of occurrences of each word available in a Data Set.

Workflow of MapReduce for word count consists of 5 steps which is shown in Figure 5.

1. **Splitting** – The splitting parameter can be anything, e.g. splitting by space, comma, semicolon, or even by a new line ('\n').
2. **Mapping** – as explained above
3. **Intermediate splitting** – the entire process in parallel on different clusters. In order to group them in "Reduce Phase" the similar KEY data should be on same cluster.
4. **Reduce** – it is nothing but mostly group by phase
5. **Combining** – The last phase where all the data (individual result set from each cluster) is combine together to form a Result

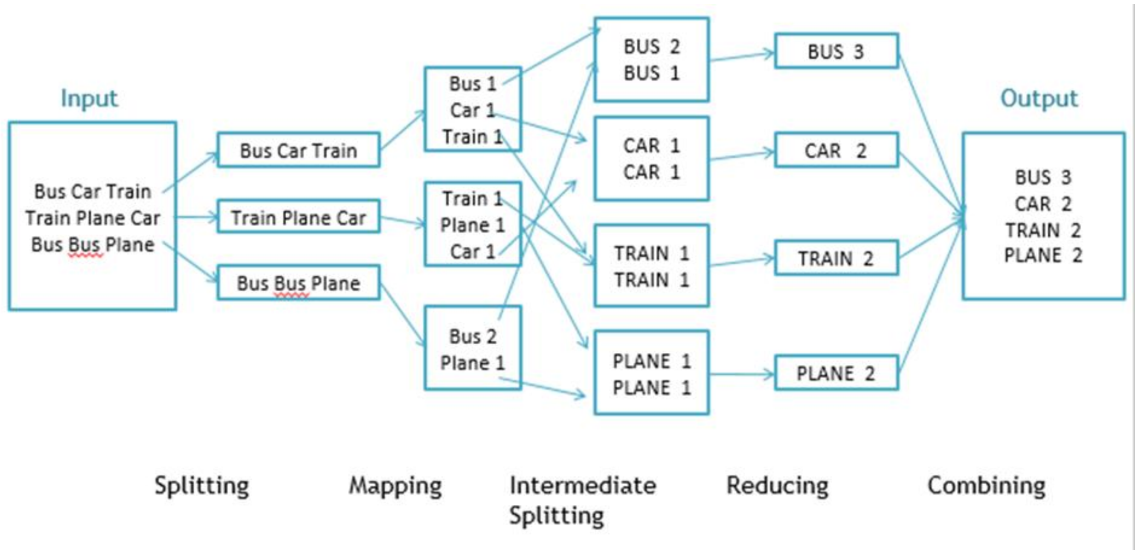


Figure 5 Work Flow Diagram for Word Count

Steps

Step 1. Open Eclipse> File > New > Java Project >(Name it – MRProgramsDemo) > Finish

Step 2. Right Click > New > Package (Name it - PackageDemo) > Finish

Step 3. Right Click on Package > New > Class (Name it - WordCount)

Step 4. Add Following Reference Libraries –

Right Click on Project > Build Path> Add External Libraries

- */usr/lib/hadoop-0.20/***hadoop-core.jar**
- *Usr/lib/hadoop-0.20/lib/***Commons-cli-1.2.jar**

Step 5. Type following Program :

```
package PackageDemo;
import java.io.IOException;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.GenericOptionsParser;
public class WordCount {
public static void main(String [] args) throws Exception
{
Configuration c=new Configuration();
String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
Path input=new Path(files[0]);
Path output=new Path(files[1]);
Job j=new Job(c,"wordcount");
j.setJarByClass(WordCount.class);
j.setMapperClass(MapForWordCount.class);
j.setReducerClass(ReduceForWordCount.class);
j.setOutputKeyClass(Text.class);
j.setOutputValueClass(IntWritable.class);
FileInputFormat.addInputPath(j, input);
FileOutputFormat.setOutputPath(j, output);
System.exit(j.waitForCompletion(true)?0:1);
}
public static class MapForWordCount extends Mapper<LongWritable, Text,
      Text, IntWritable>{
```

```
public void map(LongWritable key, Text value, Context con) throws
      IOException, InterruptedException
{
String line = value.toString();
String[] words=line.split(",");
for(String word: words )
{
  Text outputKey = new Text(word.toUpperCase().trim());
  IntWritable outputValue = new IntWritable(1);
  con.write(outputKey, outputValue);
}
}
}
public static class ReduceForWordCount extends Reducer<Text,
      IntWritable, Text, IntWritable>
{
public void reduce(Text word, Iterable<IntWritable> values, Context
      con) throws IOException, InterruptedException
{
int sum = 0;
   for(IntWritable value : values)
   {
   sum += value.get();
   }
   con.write(word, new IntWritable(sum));
}
}
}
```

Explanation

The program consist of 3 classes:

- Driver class (Public void static main- the entry point)
- Map class which **extends** public class Mapper<KEYIN,VALUEIN,KEYOUT,VALUEOUT> and implements the Map function.
- Reduce class which extends public class Reducer<KEYIN,VALUEIN,KEYOUT,VALUEOUT> and implements the Reduce function.

Step 6. Make Jar File

Right Click on Project> Export> Select export destination as **Jar File** > next> Finish

- ✓ Working on Hadoop Cluster:

admin@ubuntu$ su – hduser

hduser@ubuntu:/usr/local/hadoop$start-all.sh

hduser@ubuntu:/usr/local/hadoop$jps :

All the nodes should be up and running

```
2287 TaskTracker
2149 JobTracker
1938 DataNode
2085 SecondaryNameNode
2349 Jps
1788 NameNode
```

Step 7: Copy file and move it in HDFS

```
bin/hadoop dfs -copyFromLocal /path/input_data input_directory_in_hdfs
```

Step 8 . Run Jar file

(hadoop jar jarfilename.jar packageName.ClassName PathToInputTextFile PathToOutputDirectry)

```
hadoop jar MRProgramsDemo.jar PackageDemo.WordCount wordCountFile
 MRDir1
```

Step 9: Open Result

```
hadoop fs -ls MRDir1

hadoop fs -cat MRDir1/part-r-00000
```

or

localhost:50070 -> Utilities->Browse file system

Lab Exercises

1. Modify the word count program to display the document_id word count. Consider each sentence as one document.
2. Consider Movie Lens data, the objective is to find the movies falling more than two genres. The solution is a single Map-Reduce job.
3. Given a set of webserver logs for a site, the objective here is to find the hourly web traffic to particular site. The solution is a single Map-Reduce job. The mapper reads the logline and uses a regular expression to extract the date format from the log, parses it in to a Date object using SimpleDateFormat, and extracts the hour from it. The mapper emits the (HOUR, 1) pair to the reducer, which sorts them.

Additional Exercises

1. Consider MovieLens data, the objective is to find users who are *not* "power taggers", i.e., those who have tagged less than MIN_RATINGS (25) movies. The solution is a single Map-Reduce job using a technique called Repartitioned Join or Reduce-Side join.
   The Mapper reads both the user data and ratings data. The number of columns differ in the two datasets, so it classifies the record as User or Rating and emits (userID, recordType) to the reducer. The Reducer groups the records by recordType and sums up the occurrences. If the number of rating records are less than MIN_RATINGS, then it writes out the (userID, numberOfRatings) to HDFS.

Data set :

https://drive.google.com/open?id=0B4gbuqYuod28ZmZhN3Blbi1UcWM

LAB NO: 5 **Date:**

MAP REDUCE PROGRAMMING

Objectives:

1. To work with more than one input files
2. To perform aggregation on columns using Map Reduce functions

Mutiple Input Files in MapReduce:

When there are more than input file , same number of mapper are needed to read records from input files. For instance, if there are two input files then two mapper classes are required as shown in Figure 6.

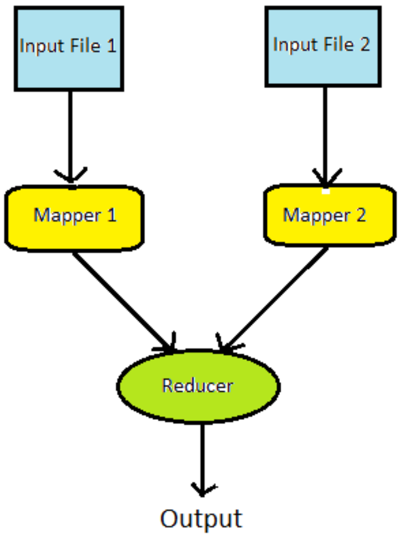


Figure 6 Multiple input files

Joining Two Files Using MultipleInput In Hadoop MapReduce - MapSide Join

.

Two different large data can be joined in map reduce programming also. Joins in Map phase refers as Map side join, while join at reduce side called as reduce side join. MapSide can be achieved using **MultipleInputFormat** in Hadoop.

Consider 2 files,One file with EmployeeID, Name, Designation and another file with EmployeeID, Salary, Department.

File1.txt
1 Anne,Admin
2 Gokul,Admin
3 Janet,Sales
4 Hari,Admin
File2.txt
1 50000,A
2 50000,B
3 60000,A
4 50000,C

Join these files into one based on **EmployeeID.** Expected results are :
1 Anne,Admin,50000,A
2 Gokul,Admin,50000,B
3 Janet,Sales,60000,A
4 Hari,Admin,50000,C

Here in both file File1.txt,File2.txt employeeId's are common. Two map jobs are required to process these file.

Processing File1.txt

```
public void map(LongWritable k, Text value, Context context) throws
 IOException, InterruptedException
{
 String line=value.toString();
 String[] words=line.split("\t");
 keyEmit.set(words[0]);
 valEmit.set(words[1]);
 context.write(keyEmit, valEmit); }
```

The above map job process File1.txt
String [] words=line.split("\t");
splits each line with \t space so words[0] will be the employeeId which we pass it as key and the rest as value.

eg: 1 Anne,Admin
words[0] = 1
words[1] = Anne,Admin

Processing File2.txt

```
public void map(LongWritable k, Text v, Context context) throws
 IOException, InterruptedException
{
 String line=v.toString();
 String[] words=line.split(" ");
 keyEmit.set(words[0]);
 valEmit.set(words[1]);
 context.write(keyEmit, valEmit);
}
```

The above map job process File2.txt

eg: 1 50000,A
words[0] = 1
words[1] = 50000,A

If the files are of same delimiter and ID comes first one can **resuse** the same map job

Common Reducer task to join the data using key.

```
String merge = "";
public void reduce(Text key, Iterable<Text> values, Context context)
{
 int i =0;
 for(Text value:values)
 {
  if(i == 0){
   merge = value.toString()+",";
  }
  else{
   merge += value.toString();
  }
  i++;
 }
 valEmit.set(merge);
 context.write(key, valEmit);
}
```

It will be caching 1 data from a mapper and appends it to string "merge".
And emit employeeId as key and merge as value.

Furnish Driver class to take 2 inputs and use **MultipleInputFormat** as InputFormat

```
public int run(String[] args) throws Exception {
 Configuration c=new Configuration();
 String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
 Path p1=new Path(files[0]);
 Path p2=new Path(files[1]);
 Path p3=new Path(files[2]);
 FileSystem fs = FileSystem.get(c);
 if(fs.exists(p3)){
  fs.delete(p3, true);
  }
 Job job = new Job(c,"Multiple Job");
 job.setJarByClass(MultipleFiles.class);
 MultipleInputs.addInputPath(job, p1, TextInputFormat.class,
MultipleMap1.class);
 MultipleInputs.addInputPath(job,p2, TextInputFormat.class,
MultipleMap2.class);
 job.setReducerClass(MultipleReducer.class);
 .
 .
}
```

```
MultipleInputs.addInputPath(job, p1, TextInputFormat.class,
 MultipleMap1.class);
MultipleInputs.addInputPath(job,p2, TextInputFormat.class,
 MultipleMap2.class);
```

p1,p2 are the Path variable holding 2 input files.

Output Expected:

```
1 Anne,Admin,50000,A
2 Gokul,Admin,50000,B
3 Janet,Sales,60000,A
4 Hari,Admin,50000,C
```

Aggregations In Hadoop MapReduce

Aggregation functions are sum, min, max, count etc. These aggregations are really useful in statistics and can be done in Hadoop MapReduce. If aggregation functions are to be done on a large data it can be done using MapReduce also. Below is the code for finding Min() and Max() for each columns of a csv file in MapReduce.

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;

public class ColumnAggregator {
 public static class ColMapper extends
   Mapper<Object, Text, Text, DoubleWritable> {
  /*   * Emits column Id as key and entire column elements as Values
 */
  public void map(Object key, Text value, Context context)
           throws IOException, InterruptedException {
   String[] cols = value.toString().split(",");
   for (int i = 0; i < cols.length; i++) {
    context.write(new Text(String.valueOf(i + 1)), new
 DoubleWritable(Double.parseDouble(cols[i])));
   }
 }
```

```
  }

  public static class ColReducer extends
    Reducer<Text, DoubleWritable, Text, DoubleWritable> {
   /*    * Reducer finds min and max of each column    */

   public void reduce(Text key, Iterable<DoubleWritable> values,
     Context context) throws IOException, InterruptedException {
    double min = Integer.MAX_VALUE, max = 0;
    Iterator<DoubleWritable> iterator = values.iterator(); //Iterating
    while (iterator.hasNext()) {
     double value = iterator.next().get();
     if (value < min) { //Finding min value
      min = value;
     }
     if (value > max) { //Finding max value
      max = value;
     }
    }
    context.write(new Text(key), new DoubleWritable(min));
    context.write(new Text(key), new DoubleWritable(max));
   }
  }
  public static void main(String[] args) throws Exception {
   Configuration conf = new Configuration();
   Job job = new Job(conf, "Min and Max");
   job.setJarByClass(ColumnAggregator.class);
   FileSystem fs = FileSystem.get(conf);
   if (fs.exists(new Path(args[1]))) {
    fs.delete(new Path(args[1]), true);
   }
   job.setOutputKeyClass(Text.class);
   job.setOutputValueClass(DoubleWritable.class);
   job.setMapperClass(ColMapper.class);
   job.setReducerClass(ColReducer.class);
   job.setInputFormatClass(TextInputFormat.class);
   job.setOutputFormatClass(TextOutputFormat.class);
   FileInputFormat.addInputPath(job, new Path(args[0]));
   FileOutputFormat.setOutputPath(job, new Path(args[1]));
   System.exit(job.waitForCompletion(true) ? 0 : 1);
  }
 }
```

Lab Exercises:

Consider the user and tweet data, each line in the user collection contains: login, name and state from a specific user. Each line in the collection of tweets has the tweet id, content, and a reference to the user who wrote that tweet.

1. Write a MapReduce program that computes the natural join between the two collections, using the reduce-side join approach.
2. Write MapReduce program that returns the number of tweets for each user name

Additional Exercises:

1. Write a MapReduce that returns the name of users that posted no tweets

Data set:

https://drive.google.com/open?id=0B4gbuqYuod28eXRUNGJKQW9hQ1k

LAB NO:6 **Date:**

APACHE PIG

Objectives:

1. To learn the basic concepts of PIG Architecture.

2. To perform data analytics using Apache PIG .

Apache Pig

Pig is a high level language with rich data analytics capabilities offering the power of Map Reduce in a simplified manner. Characteristics of Pig are

- Pig can operate on any data whether it has meta-data or not. It can operate on relational, nested, or unstructured data. And it can be extended to operate on data beyond files, including key/value stores, databases, etc.
- Pig is intended to be a language for parallel data processing. It is not tied to one particular parallel framework.
- Pig is designed to be easily controlled and modified by its users.
- Pig processes data quickly.

Pig Architecture

Pig Architecture is shown in Figure 1.

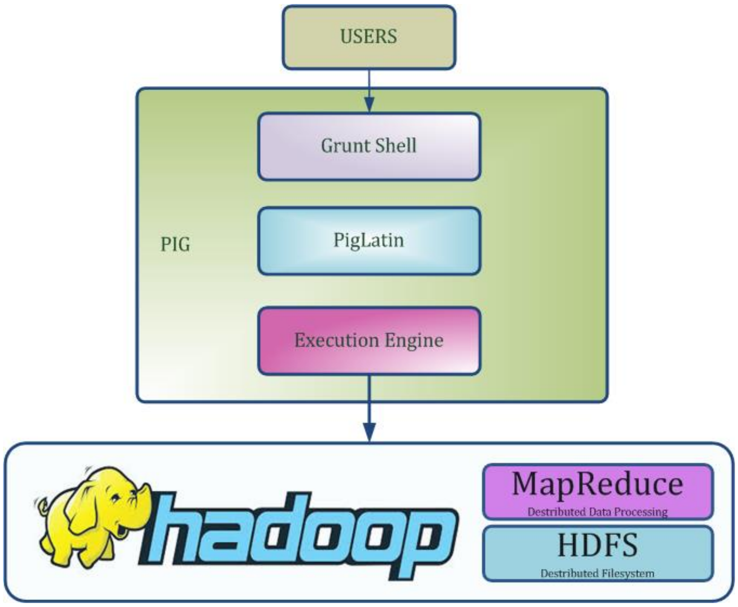


Figure 1 Apache Pig Architecture

- A grunt shell: allows for interactive command execution
- PigLatin: Data flow language. It allows users to describe how data from different inputs should be read, processed, and then stored to one or more outputs in parallel. PigLatin statements are the basic constructs to process the data.It takes relation as input and produces relation as output.
- Execution Engine: translates the PigLatin commands into MapReduce execution plans and executes them.

Running Pig

You can run Pig (execute Pig Latin statements and Pig commands) using various modes.

	Local Mode	**MapReduce Mode**
Interactive Mode	yes	yes
Batch Mode	yes	yes

Execution Modes

Pig has two execution modes or exectypes:

- **Local Mode** - To run Pig in local mode, access to a single machine is needed; all files are installed and run using your local host and file system. Specify local mode using the -x flag (pig -x local).

```
/* local mode */
$ pig -x local ...
```

- **Mapreduce Mode** - To run Pig in mapreduce mode, need access to a Hadoop cluster and HDFS installation. Mapreduce mode is the default mode; (pig OR pig -x mapreduce).

```
/* mapreduce mode */
$ pig ...
or
$ pig -x mapreduce ...
```

This example shows how to run Pig in local and mapreduce mode using the java command.

```
/* local mode */
$ java -cp pig.jar org.apache.pig.Main -x local ...


/* mapreduce mode */
$ java -cp pig.jar org.apache.pig.Main ...
or
$ java -cp pig.jar org.apache.pig.Main -x mapreduce ...
```

Interactive Mode

Pig commands can be run in interactive mode using the Grunt shell. Invoke the Grunt shell using the "pig" command and then enter Pig Latin statements and Pig commands interactively at the command line.

Example

These Pig Latin statements extract all user IDs from the /etc/passwd file. First, copy the /etc/passwd file to local working directory. Next, invoke the Grunt shell by typing the "pig" command (in local or hadoop mode). Then, enter the Pig Latin statements interactively at the grunt prompt (be sure to include the semicolon after each statement). The DUMP operator will display the results to terminal screen.

```
grunt> A = load 'passwd' using PigStorage(':');
grunt> B = foreach A generate $0 as id;
grunt> dump B;
```

Batch Mode

Pig scripts can be run in in batch mode using the "pig" command (in local or hadoop mode).

Example

The Pig Latin statements in the Pig script (id.pig) extract all user IDs from the /etc/passwd file. First, copy the /etc/passwd file to local working directory. Next, run the Pig script from the command line (using local or mapreduce mode). The STORE operator will write the results to a file (id.out).

```
/* id.pig */

A = load 'passwd' using PigStorage(':');  -- load the passwd file
B = foreach A generate $0 as id;  -- extract the user IDs
store B into ‘id.out’;  -- write the results to a file name id.out
```

Local Mode

```
$ pig -x local id.pig
```

Mapreduce Mode

```
$ pig id.pig
or
$ pig -x mapreduce id.pig
```

Pig Scripts

Use Pig scripts to place Pig Latin statements and Pig commands in a single file using the *.pig extension.

Comments in Scripts

- For multi-line comments use /* …. */
- For single-line comments use –

Pig Latin Statements

Pig Latin statements are the basic constructs you use to process data using Pig. A Pig Latin statement is an operator that takes a relation as input and produces another relation as output. (This definition applies to all Pig Latin operators except LOAD and STORE which read data from and write data to the file system.) Pig Latin statements may include expressions and schemas. Pig Latin statements can span multiple lines and must end with a semi-colon (;). By default, Pig Latin statements are processed using multi-query execution.

Pig Latin statements are generally organized as follows:

- A LOAD statement to read data from the file system.
- A series of "transformation" statements to process the data.
- A DUMP statement to view results or a STORE statement to save the results.
- Note that a DUMP or STORE statement is required to generate output.

In this example Pig will validate, but not execute, the LOAD and FOREACH statements.

```
A = LOAD 'student' USING PigStorage() AS (name:chararray, age:int, gpa:float);
B = FOREACH A GENERATE name;
```

In this example, Pig will validate and then execute the LOAD, FOREACH, and DUMP statements.

```
A = LOAD 'student' USING PigStorage() AS (name:chararray, age:int, gpa:float);
B = FOREACH A GENERATE name;
DUMP B;
(John)
(Mary)
(Bill)
(Joe)
```

Loading Data

Use the LOAD operator and the load/store functions to read data into Pig (PigStorage is the default load function).

Working with Data

Pig allows you to transform data in many ways. Some of the basic operators:

- Use the FILTER operator to work with tuples or rows of data. Use the FOREACH operator to work with columns of data.
- Use the GROUP operator to group data in a single relation. Use the COGROUP, inner JOIN, and outer JOIN operators to group or join data in two or more relations.

- Use the UNION operator to merge the contents of two or more relations. Use the SPLIT operator to partition the contents of a relation into multiple relations.

Storing Intermediate Results

Pig stores the intermediate data generated between MapReduce jobs in a temporary location on HDFS. This location must already exist on HDFS prior to use. This location can be configured using the pig.temp.dir property.

Storing Final Results

Use the STORE operator and the load/store functions to write results to the file system (PigStorage is the default store function).

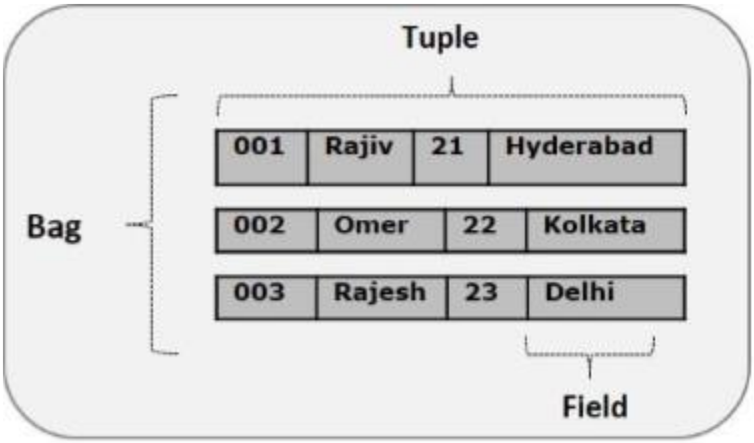
Debugging Pig Latin

Pig Latin provides operators that can help you debug your Pig Latin statements:

- Use the DUMP operator to display results to your terminal screen.
- Use the DESCRIBE operator to review the schema of a relation.
- Use the EXPLAIN operator to view the logical, physical, or map reduce execution plans to compute a relation.
- Use the ILLUSTRATE operator to view the step-by-step execution of a series of statements.

Pig Latin Data Model

The data model of Pig Latin is fully nested and it allows complex non-atomic datatypes such as **map** and **tuple**. Given below is the diagrammatical representation of Pig Latin's data model.


Tuple
Bag
001 | Rajiv | 21 | Hyderabad
002 | Omer | 22 | Kolkata
003 | Rajesh | 23 | Delhi
Field

Atom

Any single value in Pig Latin, irrespective of their data, type is known as an **Atom**. It is stored as string and can be used as string and number. int, long, float, double, chararray, and bytearray are the atomic values of Pig. A piece of data or a simple atomic value is known as a **field**.

Example − ‘raja’ or ‘30’

Tuple

A record that is formed by an ordered set of fields is known as a tuple, the fields can be of any type. A tuple is similar to a row in a table of RDBMS.

Example − (Raja, 30)

Bag

A bag is an unordered set of tuples. In other words, a collection of tuples (non-unique) is known as a bag. Each tuple can have any number of fields (flexible schema). A bag is represented by ‘{}’. It is similar to a table in RDBMS, but unlike a table in RDBMS, it is not necessary that every tuple contain the same number of fields or that the fields in the same position (column) have the same type.

Example − {(Raja, 30), (Mohammad, 45)}

A bag can be a field in a relation; in that context, it is known as **inner bag**.

Example − {Raja, 30, **{9848022338, raja@gmail.com,}**}

Map

A map (or data map) is a set of key-value pairs. The **key** needs to be of type chararray and should be unique. The **value** might be of any type. It is represented by '[]'

Example − [name#Raja, age#30]

Relation

A relation is a bag of tuples. The relations in Pig Latin are unordered (there is no guarantee that tuples are processed in any particular order).

Solved Exercise:

1. Counting the number of words in a given file sample.txt

```
Lines=LOAD 'path/sample.txt' AS (line: chararray); Words = FOREACH Lines GENERATE
    FLATTEN (TOKENIZE (line)) AS word;
Groups = GROUP Words BY word;
Counts = FOREACH Groups GENERATE group, COUNT (Words);
Results = ORDER Words BY Counts DESC;
Top5 = LIMIT Results 5;
STORE Top5 INTO /output/top5words;
```

2. Given drivers.csv and timesheet.csv compute the sum of hours and miles logged driven by a truck driver for an year

Driver.csv has the data [driverId, name,ssn,location,certified,wage-plan]

Timesheet.csv has the data [driverId, week,hours-logged,miles-logged]

```
drivers = LOAD 'drivers.csv' USING PigStorage(',');
raw_drivers = FILTER drivers BY $0>1;
drivers_details = FOREACH raw_drivers GENERATE $0 AS driverId, $1 AS name;
```

```
timesheet = LOAD 'timesheet.csv' USING PigStorage(',');
raw_timesheet = FILTER timesheet by $0>1;
timesheet_logged = FOREACH raw_timesheet GENERATE $0 AS driverId, $2 AS
    hours_logged, $3 AS miles_logged;
grp_logged = GROUP timesheet_logged by driverId;
sum_logged = FOREACH grp_logged GENERATE group as driverId,
SUM(timesheet_logged.hours_logged) as sum_hourslogged,
SUM(timesheet_logged.miles_logged) as sum_mileslogged;
join_sum_logged = JOIN sum_logged by driverId, drivers_details by driverId;
join_data = FOREACH join_sum_logged GENERATE $0 as driverId, $4 as name, $1
    as hours_logged, $2 as miles_logged;
dump join_data;
```

Lab Exercise

Consider the user and tweet data, each line in the user collection contains: login, name and state from a specific user. Each line in the collection of tweets has the tweet id, content, and a reference to the user who wrote that tweet.

1. Write a Pig Latin query that outputs the login of all users in NY State.
2. Write a Pig Latin query that returns all the tweets that include the word 'favorite', ordered by tweet id.
3. Write a Pig Latin query that returns the number of tweets for each user name (not login). You should output one user per line, in the following format:

    user_name, number_of_tweets

4. Write a Pig Latin query that returns the number of tweets for each user name (not login), ordered from most active to least active users. You should output one user per line, in the following format:

user_name, number_of_tweets

5. Write a Pig Latin query that returns the name of users that posted at least two tweets. You should output one user name per line.
6. Write a Pig Latin query that returns the name of users that posted no tweets. You should output one user name per line.

Additional Exercises

Use Pig to solve the analytic problems for a set of linked tables. There are three data files with the following schema: customer (cid, name, age, city, sex), book (isbn, name), and purchase (year, cid, isbn, seller, price), where purchase.cid is the foreign key to customer.cid and purchase.isbn is the foreign key to book.isbn. The fields in the dataset are separated by "\t".
Write pig scripts to answer the following queries. The pig local mode is recommended for better performance

1. How much did each seller earn?
2. How much did each family spend on books? Assume all users with the same last name are from the same family
3. Find the names of the books that Amazon gives the lowest price among all sellers (Exclude the case that Amazon is the only seller; it counts that Amazon and other sellers give the same price).
4. Who also bought ALL the books that Harry bought?

Data set ;
https://drive.google.com/open?id=0B4gbuqYuod28UmM0bXROc19BcEE

LAB NO: 7 **Date:**

APACHE HIVE

Objectives:

1. To understand basics of hive architecture

2. To perform data analytics using Hive

Apache Hive is a data warehouse system on Hadoop. Hive is an construct over Apache Hadoop and provides an SQL like query interface. If the user is comfortable with query languages rather than scripting or writing pure MapReduce code they can use hive to process and query data. The user only sees table like data and can use hive queries to get results from the data. Hive internally creates, plans and executes MapReduce Jobs and gives the desired results. Hive is suitable for both unstructured and semi structured data.It uses its own query language HiveQL which is similar to the traditional SQL.

Hive Architecture is shown in Figure 1. Its components are:

- Hadoop - Base Framework for hive to operate.

- Driver - To communicate with the Hadoop World.

- Command Line Interface (CLI) – The Hive CLI is the console for writing and executing Hive Queries. It can be used for operating on our data.

- Web Interface - To monitor/administrate Hive jobs.

- Metastore –It is data warehouse which stores all the structure information of various tables/partitions in Hive.

- Thrift Server – To expose Hive as a service which can then be used for connecting via JDBC/ODBC etc.

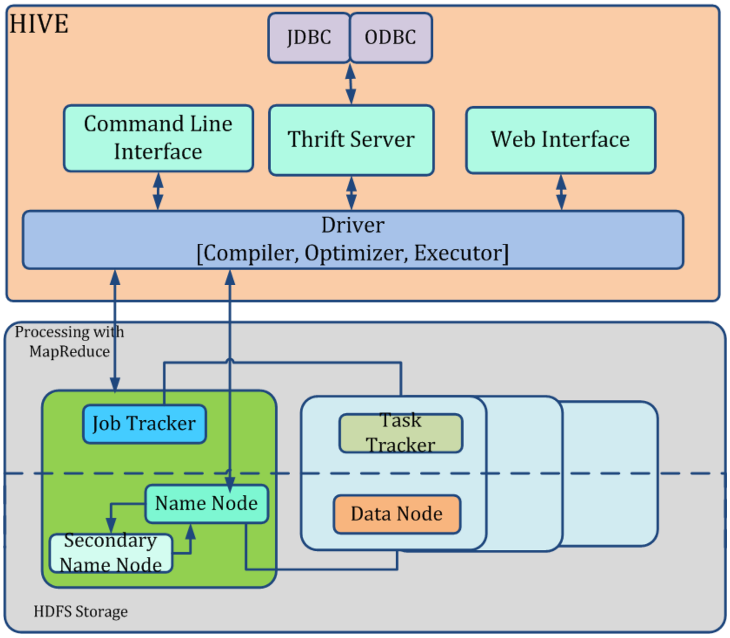


Figure 2 Hive Architecture

The components responsible for the query execution are:

- Hive Compiler: is responsible for the semantic analysis of the hive query and creating an execution plan. The execution plan is a DAG of stages.

- Hive Execution Engine: executes the execution plan created by the Hive Compiler.

- Hive Optimizer: optimizes the execution plan.

Hive can be organized into 3 data models:

- Tables – are similar to tables in relational database.

- Partitions – every table can have one or more partition keys. The data is stored in files based on the partition key specified. Without a partition all the values would be submitted to the MR Job, whereas on specifying the partition key only a small subset of data would be passed to the MR jobs. Hive makes different directories for different partitions to hold data.

- Buckets – the data in each partition may again be divided into buckets based on the hash values. Each bucket is stored as a file in the partition directory.

Hive uses Hive Query Language (HiveQL) which is similar to SQL and user can use SQL syntax for loading, querying tables. Hive queries are syntax checked by the hive compiler for rightness and execution plan is created from these queries. The Hive Executer then runs the execution plan.

When importing any new data into Hive, there is generally a three-stage process:
1. Create the specification of the table into which the data is to be imported.
2. Import the data into the created table.
3. Execute HiveQL queries against the table.

Hive Data Types

Based on the need, Hive supports primitive and complex data types as described below:

- **Primitive types**:
  - INTEGERS
    - TINY INT 1 byte integer
    - SMALL INT 2 byte integer
    - INT 4 byte integer
    - BIGINT 8 byte integer
  - BOOLEAN
    - BOOLEAN TRUE or FALSE
  - FLOATING POINT numbers
    - FLOAT Single precision
    - DOUBLE Double precision
  - STRING type
    - STRING Sequence of characters
- **Complex Types**: Complex types can be constructed using primitive data types and other composite types with the help of:
  - Structs
  - Maps or key value pairs
  - Arrays – Indexed lists

Hive Scripting

Hive scripts are used to execute a set of Hive commands collectively. Hive scripting helps us to reduce the time and effort invested in writing and executing the individual commands manually.

Writing Hive Scripts

First, open a terminal and enter the following command below to create a Hive Script sample.sql.

```
Command: gedit sample.sql
```

Hive script file should be saved with .sql extension. This will enable the execution of the commands. Now open the file in Edit mode and write your Hive commands that will be executed using this script. In this sample script, we will do the following tasks sequentially (create, describe and then load the data into the table. And then retrieve the data from table).

- Create a table 'product' in Hive:

Command:

```
create table product_dtl ( product_id: int, product-name: string, product_price: float,
product_category: string) rows format delimited fields terminated by ',' ;
```

Here { product_id, product-name, product_price, product_category} are names of the columns in the 'product_dtl' table. "Fields terminated by ',' " indicates that the columns in the input file are separated by the ',' delimiter. You can also use other delimiters as per your requirement. For example, we can consider the records in an input file separated by a new line ('\n') character.

- Describe the Table:

```
Command: describe product_dtl;
```

- Load the data into the Table:

First create an input file that contains the records that needs to be inserted into the table.

```
Command: sudo gedit input.txt
```

Input file will look like:

1. Laptop, 45000, Computers
2. Pencils, 2, Stationery
3. Rice, 64.45, Grocery
4. Furniture, 65000, Interiors

To load the data from this file we need to execute the following:

Command:

```
load data local inpath '/path/input.txt' into table product_dtl;
```

Retrieving the Data

To retrieve the data we use the simple select statement as under

Command:

```
select * from product_dtl;
```

Save this sample.sql file and run the following command

Command:

```
Hive –f /path/sample.sql
```

While executing the script, mention the entire path of the script location. Here the sample script is present in the current directory

The following output shows that the table is created and the data from our sample input file is stored in the database.

1	Laptop	45000	Computers
2	Pencils	2	Stationery
3	Rice	64.45	Groceries
4	Furniture	65000	Interiors

Lab Exercise

MovieLens data sets were collected by the GroupLens Research Project at the University of Minnesota. It represent users' reviews of movies.

This data set consists of:

- 100,000 ratings (1-5) from 943 users on 1682 movies.
- Each user has rated at least 20 movies.
- Simple demographic info for the users (age, gender, occupation, zip)

u.data -- The full u data set, 100000 ratings by 943 users on 1682 items. Each user has rated at least 20 movies. Users and items are numbered consecutively from 1. The data is randomly ordered. This is a tab separated list of user id | item id | rating | timestamp. The time stamps are unix seconds since 1/1/1970 UTC

u.user -- Demographic information about the users; this is a tab separated list of user id | age | gender | occupation | zip code The user ids are the ones used in the u.data data set.

Write Hive script to

1. Create a u_data table
2. View the field descriptions of u_data table
3. Load data into u_data table from a local text file
4. Show all the data in the newly created u_data table

5. Show the numbers of item reviewed by each user in the newly created u_data table
6. Show the numbers of users reviewed each item in the newly created u_data table
7. Create a u_user table
8. See the field descriptions of u_user table
9. Load data into u_user table from a local text file
10. Show all the data in the newly created user table
11. Count the number of data in the u_user table
12. Count the number of data in the u_user table
13. Count the number of user in the u_user table genderwise
14. Join u_data table and u_user tables based on userid and show the top 10 results

Additional Exercise

1. Write a hive script to display movie ids which are rated above 3 by at least 10 users belonging particular profession [group by profession and display highest rated movies by each occupation]

Data set:
https://drive.google.com/open?id=0B4gbuqYuod28dEpjaTF4YjhXUkU

LAB NO: 8 **Date:**

MongoDB : NOSQL

Objectives:

1. To recall basic commands of MongoDB
2. To relate RDBMS and MongoDB

Introduction

MongoDB is an Open Source database written in C++. It is open source cross platform document oriented database. Dats is stored in the form of document which is the usint of storing data. *document* use JSON style for storing data. A simple example of a JSON document is as follows :

```
{ name : "Rama" }
```

RDBMS terminology which are analogous to the MongoDB terms are as shown in the **Table 5 RDBMS vs MongoDB terminology.**

Table 6 RDBMS vs MongoDB terminology

RDBMS	**MongoDB**
Table	Collection
Column	Key
Value	Value
Records / Rows	Document / Object

A collection may store number of documents which has same structure or different structure because of schema free structure of the database.

Number of databases can be run on a single MongoDB server. Default database of MongoDB is ‘db’.

To display all the databases

```
> show dbs
```

To refer the current database object or connection

```
> db
```

To connect to a particular database

```
> use db_name
```

MongoDB Query and Projection Operators

To combine/compare multiple queries MongoDB provides many operators.

Comparison Query operators

```
$gt, $lt, $lte, $gte, $ne, $in, $nin
```

Logical Query operators

```
$and, $not, $or, $nor
```

Element Query operators

`$exists` (matches documents that have the specified field), `$type` (selects documents if a field is of the specified type.)

Evaluation Query operators

```
$mod, $regex, $where
```

Example :

Assume the collection name is “testtable”. If requirement is to fetch the value of age more than 22, the command is

```
> db.testtable.find({age : {$gt : 22}}).pretty();
```

find() method displays the documents in a non structured format but to display the results in a formatted way, the pretty() method can be used.

If requirement is to display all the records with age = 19,20,22 and 25, in operator must be used.

```
> db.testtable.find({"age" : { $in : [19,20,22,25]}})
```

To insert a new object in testtable,

```
> db.testtable.insert({"user_id":"user5","password":"user5",
"sex":"Male","age":21, "date_of_join":"17/08/2011", "education":"MCA",
"profession":"S.W.Engineer","interest":"SPORTS","extra":
{"community_name" :["ATHELATIC", "GAMES FAN GYES","FAVOURIT
GAMES"]}});
```

To update an existing object, update() is useful. It takes 4 arguments. Criteria, objectnew, upsert and multi. Criteria is select appropriate record to update, objectnew specifies the updated information, upsert do update the if match the criteria and insert a record if not

Assume the collection name is user details and the data of user_id has to updated,

```
> db.userdetails.update({"user_id" : "QRSTBWN"},{"user_id" :
"QRSTBWN","password" :"NEWPASSWORD" ,"date_of_join" : "17/10/2010"
,"education" :"M.B.A." , "profession" : "MARKETING","interest" :
"MUSIC","community_name" :["MODERN MUSIC", "CLASSICAL MUSIC","WESTERN
MUSIC"],"community_moder_id" : ["MR. BBB","MR. JJJ","MR
MMM"],"community_members" : [500,200,1500],"friends_id" :
["MMM123","NNN123","OOO123"],"ban_friends_id"
:["BAN123","BAN456","BAN789"]});
```

To remove a specific object

```
   > db.userdetails.remove( { "user_id" : "testuser" } )
```

To remove all data from collection

```
   > db.userdetails.remove({})
```

To delete entire collection

```
> db.userdetails.drop()
```

Remove an entire Database

```
> db.dropDatabase()
```

Example :
Consider the dataset restaurant

1. Display the fields restaurant_id, name, borough and suisine for all the documents in the collection restaurant.

```
db.restaurants.find({},{"restaurant_id":1,"name":1,"borough":1,"cui
sine" :1});
```

2. Write a MongoDB query to display the fields restaurant_id, name, borough and zip code, but exclude the field _id for all the documents in the collection restaurant.

```
db.restaurants.find({},{"restaurant_id" :
1,"name":1,"borough":1,"address.zipcode" :1,"_id":0});
```

3. Write a MongoDB query to find the restaurants that achieved a score is more than 80 but less than 100.

```
db.restaurants.find({grades : { $elemMatch:{"score":{$gt : 80 , $lt
:100}}}});
```

4. Write a MongoDB query to find the restaurants that do not prepare any cuisine of 'American' and their grade score more than 70 and latitude less than -65.754168.

```
db.restaurants.find(
               {$and:
                    [
                       {"cuisine" : {$ne :"American "}},
                       {"grades.score" : {$gt : 70}},
                       {"address.coord" : {$lt : -65.754168}}
                    ]
                }
```

```
                    );
```

5. Write a MongoDB query to find the restaurants which belong to the borough Bronx and prepared either American or Chinese dish.

```
db.restaurants.find(
{
"borough": "Bronx" ,
$or : [
{ "cuisine" : "American " },
{ "cuisine" : "Chinese" }
]
}
);
```

6. Write a MongoDB query to find the restaurant name, borough, longitude and attitude and cuisine for those restaurants which contain 'Mad' as first three letters of its name.

```
db.restaurants.find(
                   { name :
                     { $regex : /^Mad/i, }
                   },
                       {
                         "name":1,
                         "borough":1,
                         "address.coord":1,
                         "cuisine" :1
                        }
                   );
```

Lab Exercise

1. Consider the dataset World_Bank. Write the mongodb queries for the following.
   - i. Display the fields “_id” and “"mjthemecode”
   - ii. Display all the data related to “Republic of India”
   - iii. Display the data where sector_namecode is “Water Supply”

2. Consider the dataset Vehicle_registration
   https://data.gov.in/node/2865481/datastore/export/json

The dataset consists of total vehicle registered in India state wise. Write query for the following.

  i. How many vehicles have been registered in state Karnataka?

ii. How many taxis and buses have been registered in state Karnataka?
iii. How many omnibuses or cars have been registered in state Manipura?

References

1. http://www.w3resource.com/mongodb-exercises/
2. https://docs.mongodb.com/manual/tutorial/
3. https://data.gov.in/catalog/

Dataset

To download the dataset,
https://drive.google.com/open?id=0B4gbuqYuod28RlFGb2JnNTZYc2c

LAB NO: 9 **Date:**

PROJECT SYNOPSIS

Objectives:

In this lab students should be able to:

1. Submit the synopsis of the project that they are supposed to implement using any data mining algorithm(s).
2. Understand the basic concepts of algorithm(s) chosen and submit reference papers related to the existing system.

Format for the Project Synopsis

Title of the project

Date : <place the date of submission here>

Introduction

<Introduction should contain the introduction to the domain and brief description about the project work. It should reveal the context in which you have made your project. It should talk about why you are interested in proposing such a solution.>

Problem Definition

< Problem definition should furnish about the problems faced without such a solution as proposed by you>

Objectives

<Should mention, as a developer what is that you are doing in order to obtain a complete solution to the problem. Put together a few objectives pointwise>

Methodology

< Should mention, as a developer how are you going to give the solution to the objective. Include a framework of your work. >

Hardware/Software requirements

<Mention the names of hardware and software used to accomplish the project>

Submitted by

Name registration no

Work division:

LAB NO: 10 **Date:**

ANALYSIS AND DESIGN OF THE PROJECT

Objectives:

1. In this lab, student should submit design diagrams for their project and present the project implemented so far.

Lab Exercise

1. Draw the diagrams relevant for the project
2. Show the demonstration of project implemented so far.

LAB NO: 11 **Date:**

IMPLEMENTATION OF THE PROJECT

Objectives:

1. In this lab, student will be able to implement all the requirements specified in synopsis

Lab Exercise

1. Implement the project using the data analytical tools learnt

---

LAB NO: 12 **Date:**

TESTING OF THE PROJECT

Objective

In this lab students should be able to:

1. Demonstrate a working project and explain the underlying algorithms or methods used.