1. **Explain the methods provided by R for aggregation and reshaping of data. Give examples.**

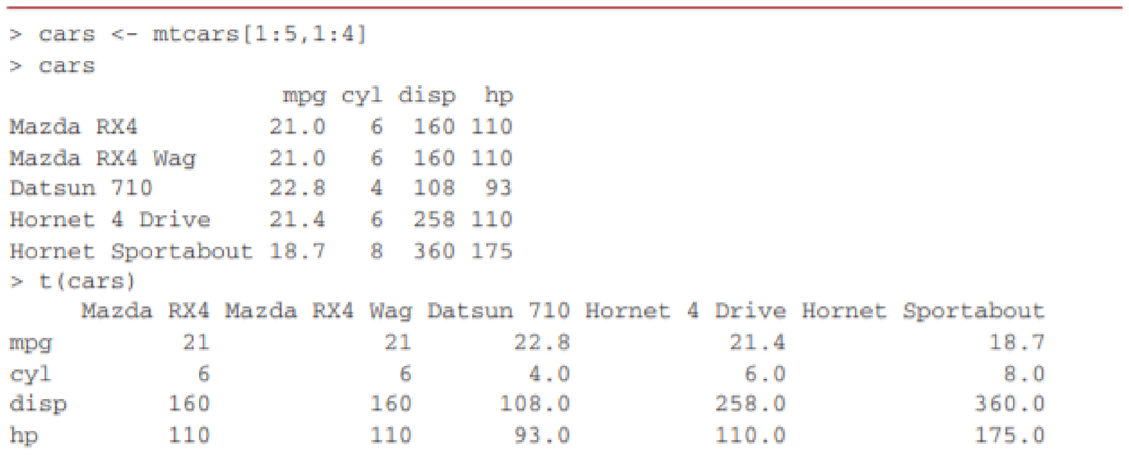
   Aggregation and restructuring

R provides a number of powerful methods for aggregating and reshaping data. When we aggregate data, we replace groups of observations with summary statistics based on those observations.

When reshaping data, we alter the structure (rows and columns) determining how the data is organized. We'll use the mtcars data frame. This dataset, describes the design and performance characteristics (number of cylinders, displacement, horsepower, mpg, and so on) of automobiles.

Transpose

The transpose (reversing rows and columns) is perhaps the simplest method of reshaping a dataset. Use the t() function to transpose a matrix or a data frame. In the latter case, row names become variable (column) names. An example is below.



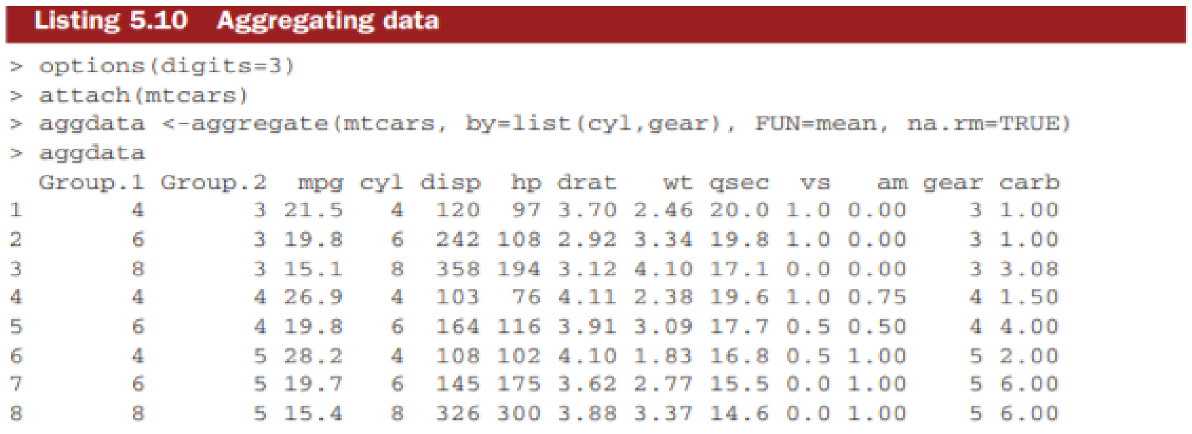
```
> cars <- mtcars[1:5,1:4]
> cars
                   mpg cyl disp  hp
Mazda RX4         21.0   6  160 110
Mazda RX4 Wag     21.0   6  160 110
Datsun 710        22.8   4  108  93
Hornet 4 Drive    21.4   6  258 110
Hornet Sportabout 18.7   8  360 175
> t(cars)
     Mazda RX4 Mazda RX4 Wag Datsun 710 Hornet 4 Drive Hornet Sportabout
mpg         21            21       22.8           21.4              18.7
cyl          6             6        4.0            6.0               8.0
disp       160           160      108.0          258.0             360.0
hp         110           110       93.0          110.0             175.0
```

Listing above uses a subset of the mtcars dataset in order to conserve space on the page.

Aggregating data

It's relatively easy to collapse data in R using one or more by variables and a defined function. The format is aggregate(x, by, FUN) where x is the data object to be collapsed, by is a list of variables that will be crossed to form the new observations, and FUN is the scalar function used to calculate summary statistics that will make up the new observation values.

As an example, we'll aggregate the mtcars data by number of cylinders and gears, returning means on each of the numeric variables.



Listing 5.10 Aggregating data

```
> options(digits=3)
> attach(mtcars)
> aggdata <-aggregate(mtcars, by=list(cyl,gear), FUN=mean, na.rm=TRUE)
> aggdata
  Group.1 Group.2  mpg cyl disp  hp drat   wt qsec  vs   am gear carb
1       4       3 21.5   4  120  97 3.70 2.46 20.0 1.0 0.00    3 1.00
2       6       3 19.8   6  242 108 2.92 3.34 19.8 1.0 0.00    3 1.00
3       8       3 15.1   8  358 194 3.12 4.10 17.1 0.0 0.00    3 3.08
4       4       4 26.9   4  103  76 4.11 2.38 19.6 1.0 0.75    4 1.50
5       6       4 19.8   6  164 116 3.91 3.09 17.7 0.5 0.50    4 4.00
6       4       5 28.2   4  108 102 4.10 1.83 16.8 0.5 1.00    5 2.00
7       6       5 19.7   6  145 175 3.62 2.77 15.5 0.0 1.00    5 6.00
8       8       5 15.4   8  326 300 3.88 3.37 14.6 0.0 1.00    5 6.00
```

In these results, Group.1 represents the number of cylinders (4, 6, or 8) and Group.2 represents the number of gears (3, 4, or 5). For example, cars with 4 cylinders and 3 gears have a mean of 21.5 miles per gallon (mpg).

When we're using the aggregate() function , the by variables must be in a list (even if there's only one). You can declare a custom name for the groups from within the list, for instance, using by=list(Group.cyl=cyl, Group.gears=gear) . The function specified can be any built-in or user-provided function. This gives the aggregate command a great deal of power. But when it comes to power, nothing beats the reshape package.

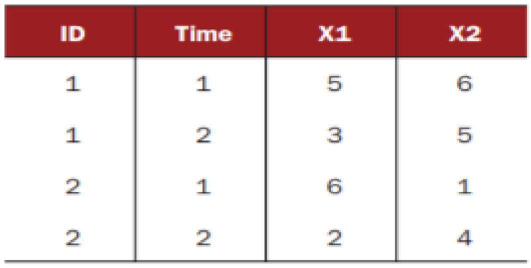
The Reshape package

The reshape package can be used for both restructuring and aggregating datasets.Reshape can be installed using **install.packages("reshape").**

Basically, we'll "melt" data so that each row is a unique ID-variable combination. Then we'll "cast" the melted data into any shape of desire. During the cast, one can aggregate the data with any function of our wish.

The dataset we'll be working with is shown in table 5. In this dataset, the measurements are the values in the last two columns (5, 6, 3, 5, 6, 1, 2, and 4). Each measurement is uniquely identified by a combination of ID variables (in this case ID, Time, and whether the measurement is on X1 or X2). For example, the measured value 5 in the first row is uniquely identified by knowing that it's from observation (ID) 1, at Time 1, and on variable X1.

Table 5.8 The original dataset (`mydata`)



ID	Time	X1	X2
1	1	5	6
1	2	3	5
2	1	6	1
2	2	2	4

MELTING

Melting a dataset means to restructure it into a format where each measured variable is in its own row, along with the ID variables needed to uniquely identify it.

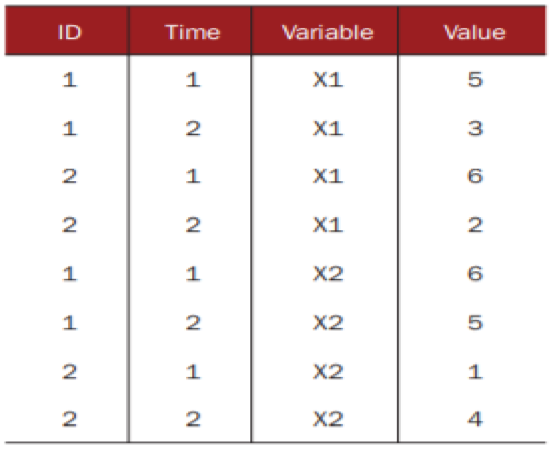
If we melt the data from table 5.8, using the following code

library(reshape)

md <- melt(mydata, id=(c("id", "time")))

we end up with the structure shown in table 5.9.

Table 5.9 The melted dataset



ID	Time	Variable	Value
1	1	X1	5
1	2	X1	3
2	1	X1	6
2	2	X1	2
1	1	X2	6
1	2	X2	5
2	1	X2	1
2	2	X2	4

We must specify the variables needed to uniquely identify each measurement (ID and Time) and that the variable indicating the measurement variable names (X1 or X2) is created automatically.

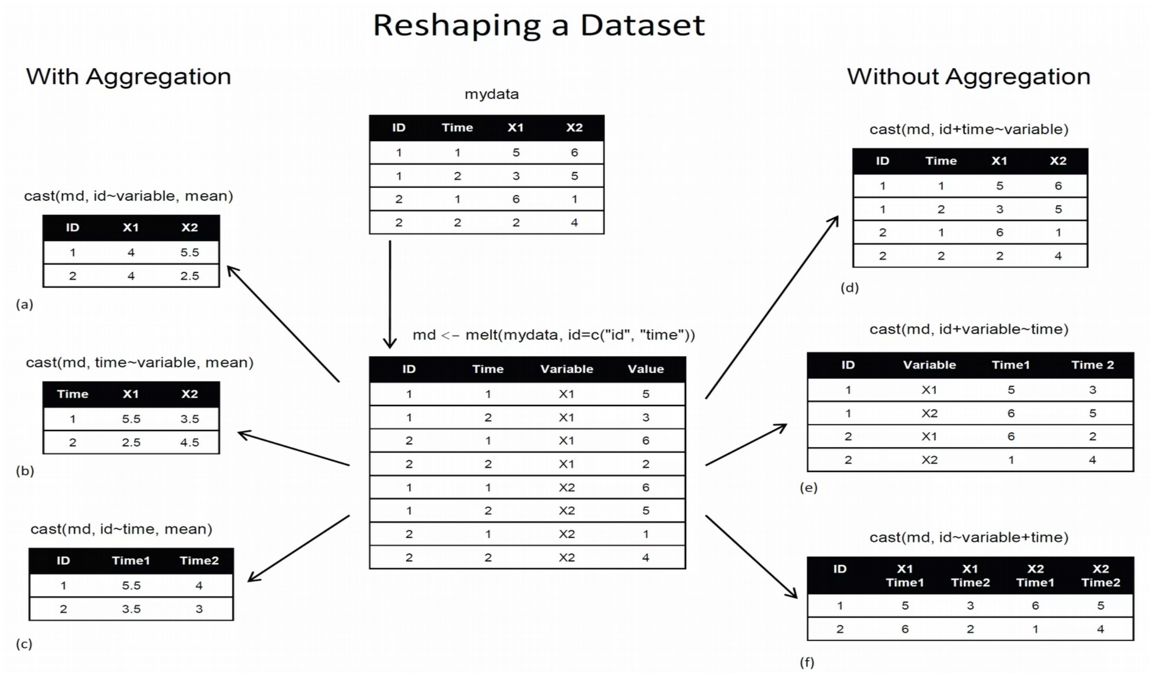
Now that we haves data in a melted form, we can recast it into any shape, using the cast() function.

CASTING

The cast() function starts with melted data and reshapes it using a formula that we provide and an (optional) function used to aggregate the data. The format is

newdata <- cast(md, formula, FUN)

where md is the melted data, formula describes the desired end result, and FUN is the (optional) aggregating function.



Reshaping a Dataset

mydata

ID	Time	X1	X2
1	1	5	6
1	2	3	5
2	1	6	1
2	2	2	4

```
md <- melt(mydata, id=c("id", "time"))
```

ID	Time	Variable	Value
1	1	X1	5
1	2	X1	3
2	1	X1	6
2	2	X1	2
1	1	X2	6
1	2	X2	5
2	1	X2	1
2	2	X2	4

With Aggregation

```
cast(md, id~variable, mean)
```

ID	X1	X2
1	4	5.5
2	4	2.5

(a)

```
cast(md, time~variable, mean)
```

Time	X1	X2
1	5.5	3.5
2	2.5	4.5

(b)

```
cast(md, id~time, mean)
```

ID	Time1	Time2
1	5.5	4
2	3.5	3

(c)

Without Aggregation

```
cast(md, id+time~variable)
```

ID	Time	X1	X2
1	1	5	6
1	2	3	5
2	1	6	1
2	2	2	4

(d)

```
cast(md, id+variable~time)
```

ID	Variable	Time1	Time 2
1	X1	5	3
1	X2	6	5
2	X1	6	2
2	X2	1	4

(e)

```
cast(md, id~variable+time)
```

ID	X1 Time1	X1 Time2	X2 Time1	X2 Time2
1	5	3	6	5
2	6	2	1	4

(f)

2. **Describe the variety of objects in R : creation, storing data, accessing the data.**

Vectors in R

- Vector is a basic data structure in R that contains element of similar type.
- These data types in R can be logical, integer, double, character, complex or raw. In R using the function, **typeof()** one can check the data type of vector. One more significant property of R vector is its length.
- The function **length()** determines the number of elements in the vector.

```
X<-C(4,5,6,7,8,9)
[1] 4 5 6 7 8 9
```

Accessing the values

```
X[2]
[1] 5
X[-4]
[1] 4 5 6 8 9 #it returns all the elements except 4th element
X[4]
[1] 7 #returns only the 4th element
```

How to Create Array in R

we will create an R array of two 3×3 matrices each with 3 rows and 3 columns.

# Create two vectors of different lengths.

```
vector1 <- c(2,9,3)
vector2 <- c(10,16,17,13,11,15)
```

Take these vectors as input to the array.

```
result <- array(c(vector1,vector2),dim = c(3,3,2))
print(result)
```

Matrices in R

Matrices are Data frames which contain lists of homogeneous data in a tabular format. We can perform arithmetic operations on some elements of the matrix or the whole matrix itself in R. Let us see how to convert a single dimension vector into a two-dimensional array using the **matrix()** function:

```
matrix(c(0,1,2,3,4,5,6,7,8,9),nrow=5,ncol=2,byrow=TRUE)

     [,1] [,2]
[1,]    0    1
[2,]    2    3
[3,]    4    5
[4,]    6    7
[5,]    8    9

matrix1<-matrix(c(0,1,2,3,4,5,6,7,8,9),nrow=5,ncol=2,byrow=TRUE)
```

Accessing the elements from the matrix

```
matrix1[3,2]
[1] 5
s
```

Lists in R

Lists are R Data Types stores collections of objects of differing lengths and types using **list()** function.

R List is the object which Contains elements of different types – like strings, numbers, vectors and another list inside it. R list can also contain a matrix or a function as its elements. The List is been created using **list()** **Function in R**. In other words, a list is a generic vector containing other objects.

For Example:
The variable x is containing copies of three vectors n, s, b and a numeric value 3.

```
1.    n = c(2, 3, 5)
2.    s = c("aa", "bb", "cc", "dd", "ee" )
3.    b = c(TRUE, FALSE, TRUE, FALSE, FALSE )
4.    x = list( n, s, b, 3) # x contains copies of n, s, b
```

n=c(4,5,7)
accessing list elements
n[2]
[1] 5

n[-1]

[1] 5 7

Array in R

In arrays, data is stored in the form of matrices, rows, and columns. We can use the matrix level, row index, and column index to access the matrix elements.

R arrays are the data objects which can store data in more than two dimensions. An array is created using the **array() function**. We can use vectors as input and create an array using the below-mentioned values in the dim parameter.

```
Array_NAME <- array(data, dim = (row_Size, column_Size, matrices, dimnames)
```

- **data** – Data is an input vector that is given to the array.
- **matrices** – Array in R consists of multi-dimensional matrices.
- **row_Size** – row_Size describes the number of row elements that an array can store.
- **column_Size** – Number of column elements that can be stored in an array.
- **dimnames** – Used to change the default names of rows and columns to the user's preference.

3. **Explain functions from ggplot2 package.**

Aesthetic Mapping

Aesthetic mappings describe how variables in the data are mapped to visual properties (aesthetics) of geoms.

- ➔ position (i.e., on the x and y axes)
- ➔ color (“outside” color)
- ➔ fill (“inside” color)
- ➔ shape (of points)
- ➔ linetype
- ➔ size

Geometic Objects (geom)

Geometric objects are the actual marks we put on a plot. Examples include:

- •points (geom_point, for scatter plots, dot plots, etc)
- •lines (geom_line, for time series, trend lines, etc)
- •boxplot (geom_boxplot, for, well, boxplots!)

A plot must have at least one geom

print.ggplot {ggplot2}
geom_polygon {ggplot2}
geom_dotplot {ggplot2}

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

There are two types of bar charts: geom_bar() and geom_col().

Points (Scatterplot)

Now that we know about geometric objects and aesthetic mapping, we can make a ggplot. geom_point requires mappings for x and y, all others are optional.

```
ggplot(housing, aes(y = Structure.Cost, x = Land.Value)) + geom_point()
```

Lines (Prediction Line)

A plot constructed with ggplot can have more than one geom. In that case the mappings established in the ggplot() call are plot defaults that can be added to or overridden. Ex. regression line

```
p1 <- ggplot(housing, aes(y = Structure.Cost, x = Land.Value)) + geom_point()
p1 + geom_point(aes(color = Home.Value)) + geom_line(aes(y = pred.SC))
```

Smoothers

Not all geometric objects are simple shapes–the smooth geom includes a line and a ribbon.

```
p1 + geom_point(aes(color = Home.Value)) + geom_smooth()
```

Text (Label Points)
Each geom accepts a particualar set of mappings;for example geom_text() accepts a labels mapping.

```
p1 + geom_text(aes(label=State), size = 3)
```

Aesthetic Mapping VS Assignment

```
p1 + geom_point(aes(size = 2), color="red")
```

4. Creation of dataset in R.

Now you have three different vectors in your workspace:

- ➔a character vector called employee, containing the names
- ➔a numeric vector called salary, containing the yearly salaries
- ➔a date vector called startdate, containing the dates on which the contracts started

```
employee <- c('John Doe','Peter Gynn','Jolie Hope')

startdate <- as.Date(c('2010-11-1','2008-3-25','2007-3-14'))

salary <- c(21000, 23400, 26800)
```

Next, you combine the three vectors into a data frame using the following code:

```
employ.data <- data.frame(employee, salary, startdate)
```

The result of this is a data frame, employ.data, with the following structure:

```
str(employ.data)

 'data.frame': 3 obs. of 3 variables:

$ employee : Factor w/ 3 levels "John Doe","Jolie Hope",..: 1 3 2

$ salary : num 21000 23400 26800

$ startdate: Date, format: "2010-11-01" "2008-03-25" ...
```

5. **Explain the advantages of map reduce mechanisms?**

Scalability - Hadoop is a platform that is highly scalable. This is largely because of its ability to store as well as distribute large data sets across plenty of servers. These servers can be inexpensive and can operate in parallel. And with each addition of servers one adds more processing power.

Cost-effective solution - Hadoop's highly scalable structure also implies that it comes across as a very cost-effective solution for businesses that need to store ever growing data dictated by today's requirements. Hadoop's scale-out architecture with MapReduce programming, allows the storage and processing of data in a very affordable manner.

Flexibility - Business organisations can make use of Hadoop MapReduce programming to have access to various new sources of data and also operate on different types of data, whether they are structured or unstructured. This allows them to generate value from all of the data that can be accessed by them.

Fast - Hadoop uses a storage method known as distributed file system, which basically implements a mapping system to locate data in a cluster. The tools used for data processing, such as MapReduce programming, are also generally located in the very same servers, which allows for faster processing of data.

Parallel processing - One of the primary aspects of the working of MapReduce programming is that it divides tasks in a manner that allows their execution in parallel.Parallel processing allows multiple processors to take on these divided tasks, such that they run entire programs in less time.

Availability and resilient nature - When data is sent to an individual node in the entire network, the very same set of data is also forwarded to the other numerous nodes that make up the network. Thus, if there is any failure that affects a particular node, there are always other copies that can still be accessed whenever the need may arise. This always assures the availability of data.

One of the biggest advantages offered by Hadoop is that of its fault tolerance. Hadoop MapReduce has the ability to quickly recognize faults that occur and then apply a quick and automatic recovery solution. This makes it a game changer when it comes to big data processing.

Simple model of programming - Among the various advantages that Hadoop MapReduce offers, one of the most important ones is that it is based on a simple programming model. This basically allows programmers to develop MapReduce programs that can handle tasks with more ease and efficiency.

The programs for MapReduce can be written using Java, which is a language that isn't very hard to pickup and is also used widespread. Thus, it is easy for people to learn and write programs that meets their data processing needs sufficiently.

6. Explain HDFS Diagram architecture with diagram

The Hadoop Distributed File System (HDFS) is designed to provide a fault-tolerant file system designed to run on commodity hardware. The primary objective of HDFS is to store data reliably even in the presence of failures including Name Node failures, Data Node failures and network partitions.

HDFS holds very large amount of data and provides easier access. To store such huge data, the files are stored across multiple machines. These files are stored in redundant fashion to rescue the system from possible data losses in case of failure. HDFS also makes applications available to parallel processing.

Features of HDFS

- ➔ It is suitable for the distributed storage and processing.
- ➔ Hadoop provides a command interface to interact with HDFS.
- ➔ The built-in servers of namenode and datanode help users to easily check the status of cluster.
- ➔ Streaming access to file system data.
- ➔ HDFS provides file permissions and authentication.

HDFS uses a master/slave architecture in which one device (the master) controls one or more other devices (the slaves). The HDFS cluster consists of a single Name Node and a master server manages the file system namespace and regulates access to files.

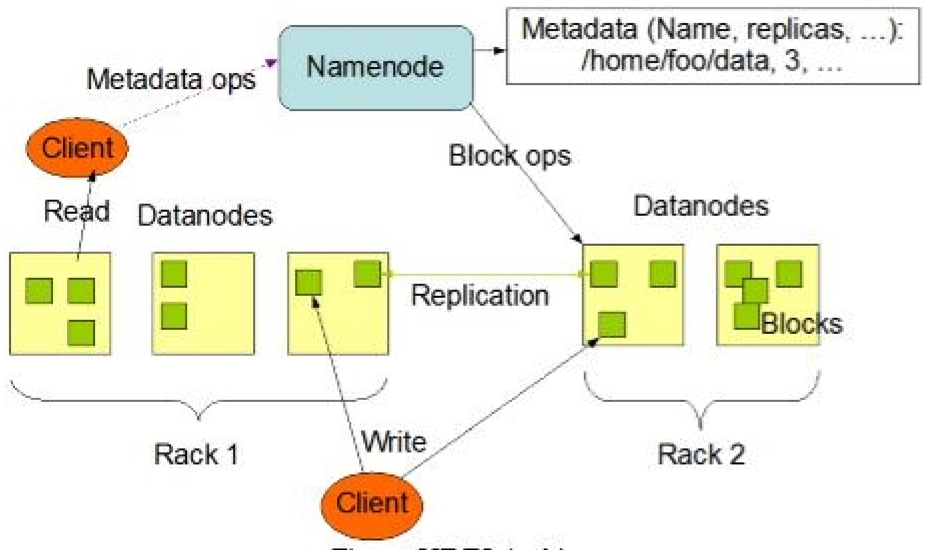


Figure: HDFS Architecture

The main components of HDFS are as described below:

Namenode

The namenode is the commodity hardware that contains the GNU/Linux operating system and the namenode software. It is a software that can be run on commodity hardware. The system having the namenode acts as the master server and it does the following tasks −

- ➔ Manages the file system namespace.
- ➔ Regulates client's access to files.
- ➔ It also executes file system operations such as renaming, closing, and opening files and directories.

Datanode

The datanode is a commodity hardware having the GNU/Linux operating system and datanode software. For every node (Commodity hardware/System) in a cluster, there will be a datanode. These nodes manage the data storage of their system.

- ➔ Datanodes perform read-write operations on the file systems, as per client request.
- ➔ They also perform operations such as block creation, deletion, and replication according to the instructions of the namenode.

Block

Generally the user data is stored in the files of HDFS. The file in a file system will be divided into one or more segments and/or stored in individual data nodes. These file segments are called as blocks.

The File System Namespace:

HDFS supports a traditional hierarchical file organization. A user or an application can create directories and store files inside these directories. The file system namespace hierarchy is similar to most other existing file systems; one can create and remove files, move a file from one directory to another, or rename a file. The NameNode maintains the file system namespace. Any change to the file system namespace or its properties is recorded by the NameNode.

Data Replication:

HDFS is designed to reliably store very large files across machines in a large cluster. It stores each file as a sequence of blocks; all blocks in a file except the last block are the same size. The blocks of a file are replicated for fault tolerance. The block size and replication factor are configurable per file. An application can specify the number of replicas of a file. The replication factor can be specified at file creation time and can be changed later.

7. Write a function in R that calculates the central tendency and spread of data objects. The function should give you a choice between parametric (mean and standard deviation) and nonparametric (median and median absolute deviation) statistics. The results should be returned as a named list. Additionally, the user should have the choice of automatically printing the results, or not. Unless otherwise specified, the function's default behavior should be to calculate parametric statistics and not print the results.

ANS)
```
list <- c(2,3,4,1,8,7,4,3,9)
para <- function(list) {
        m = mean(list)
        s = sd(mean)
        return m,s
}

non_para  <- function(list) {
        me <- median(list)
        mad <-  meadian( abs(list – meadian(list)) )
        return me, mad
}
x = para(list)
y = non_para(list)
print(x)
print(y)
```

8 Write a R function mydate() that uses the switch construct to print today's date. This function gives the user a choice regarding the format of today's date as either short or long. Example, mydate("long") prints] "Thursday December 02 2010" whereas mydate("short") prints "12-02-10". Any other type is not to be recognized. "long" is the default format for dates if type isn't specified.

```
mydate <- function(r="long"){
 switch(r,
  long={
   format(Sys.time(), "%A %B %d %Y")
  },
  short={
   format(Sys.time(), "%d-%m-%y")
  },
  {
    print('type is not recognized')
  }
 )
}
mydate("long")
mydate("short")
mydate()
```

9 Explain the creation and manipulation of data frames in R with suitable examples. Given the patientData dataset below, cross tabulate diabetes type by status.

Data Frame

A **data frame** is used for storing data tables. It is a list of vectors of equal length. For example, the following variable df is a data frame containing three vectors n, s, b.

n = c(2, 3, 5)

s = c("aa", "bb", "cc")

b = c(TRUE, FALSE, TRUE)

mtcars = *data.frame(n, s, b)*

Here is the cell value from the first row, second column of mtcars.

```
mtcars[1, 2]
[1] 6
```

Moreover, we can use the row and column names instead of the numeric coordinates.

```
mtcars["Mazda RX4", "cyl"]
[1] 6
```

Lastly, the number of data rows in the data frame is given by the **nrow** function.

```
nrow(mtcars)    # number of data rows
[1] 32
```

And the number of columns of a data frame is given by the **ncol** function.

```
ncol(mtcars)    # number of columns
[1] 11
```

Further details of the mtcars data set is available in the R documentation.

```
help(mtcars)
```

Instead of printing out the entire data frame, it is often desirable to preview it with the head function beforehand.

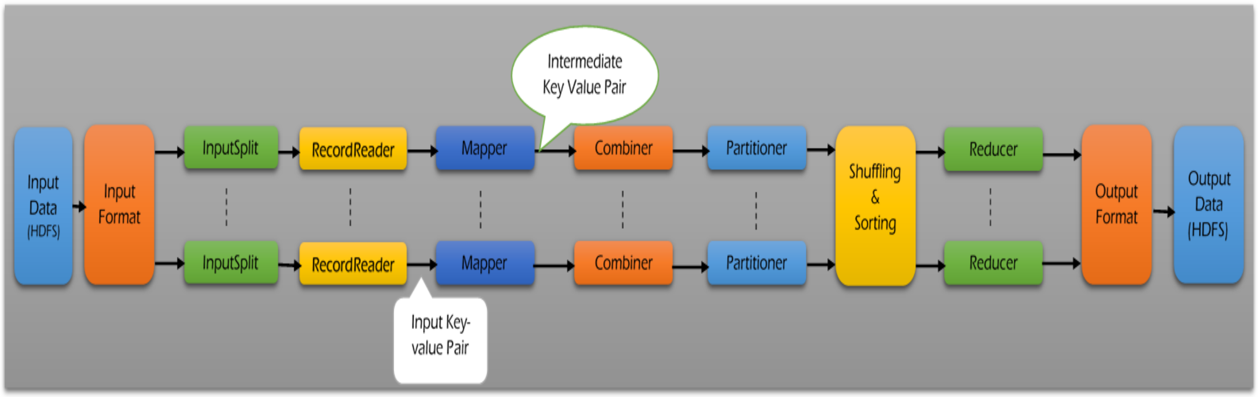
head(mtcars)

patientdata

```
patientID age  diabetes  status
1    1        25    Type1      Poor
2    2        34    Type2      Improved
3    3        28    Type1      Excellent
4    4       52    Type1      Poor
```

```
patientID <-c(1,2,3,4)
age<-c(25,34,28,52)
diabetes<-c("type1","type2","type1","type1")
status<-c("poor","improved","excellent","poor")
data <- data.frame(patientID,age,diabetes,status)
tab <- table(data$diabetes, data$status)
tab1<- crosstab(data, row.vars = "diabetes", col.vars = "status")
```

11 Explain the architecture of MapReduce.



MapReduce is a programming model and expectation is parallel processing in Hadoop. MapReduce makes easy to distribute tasks across nodes and performs Sort or Merge based on distributed computing.

The underlying system takes care of partitioning input data, scheduling the programs execution across several machines, handling machine failures and managing inter-machine communication.

Input will be divided into multiple chunks/blocks.

Each and every chunk/block of data will be processed in different nodes.

MapReduce architecture contains the below phases -

1.Input Files

2.InputFormat

3.InputSplit

4.RecordReader

5.Mapper

6.Combiner

7.Partitioner

8.Shuffling and Sorting

9.Reducer

10.RecordWriter

11.OutputFormat

Input Files: -

➔ In general, the input data to process using MapReduce task is stored in input files.

➔ These input files typically reside in HDFS (Hadoop Distributed File System).

➔ The format of these files is random where other formats like binary or log files can also be used.

InputFormat: -

➔ InputFormat describes the input-specification for a Map-Reduce job.

➔ InputFormat defines how the input files are to split and read.

➔ InputFormat selects the files or other objects used for input.

➔ InputFormat creates InputSplit from the selected input files.

➔ InputFormat split the input into logical InputSplits based on the total size, in bytes of the input files.

InputSplit: -

➔ InputSplit is created by InputFormat.

➔ InputSplit logically represents the data to be processed by an individual Mapper.

➔ One map task is created to process one InputSplit.

➔ The number of map tasks normally equals to the number of InputSplits.

➔ The InputSplit is divided into input records and each record is processed by the specific mapper assigned to process the InputSplit.

➔ InputSplit presents a byte-oriented view on the input.

RecordReader: -

➔ RecordReader communicates with the InputSplit in Hadoop MapReduce.

➔ RecordReader reads

➔ RecordReader converts the byte-oriented view of the input from the InputSplit.

➔ RecordReader provides a record-oriented view of the input data for mapper and reducer tasks processing.

➔ RecordReader converts the data into key-value pairs suitable for reading by the mapper.

➔ RecordReader communicates with the InputSplit until the file reading is not completed.

➔ Once the file reading completed, these key-value pairs are sent to the mapper for further processing.

Mapper: -

➔ Mapper processes each input record and generates new key-value pair.

➔ Mapper generated key-value pair is completely different from the input key-value pair.

➔ The mapper output is called as intermediate output.

➔ The mapper output is not written to local disk because of it creates unnecessary copies.

➔ Mappers output is passed to the combiner for further process.

➔ Map takes a set of data and converts it into another set of data, where individual elements are broken down into key pairs.

➔ The Mapper reads the data in the form of key/value pairs and outputs zero or more key/value pairs.

Combiner: -

➔ Combiner acts as a mini reducer in MapReduce framework.

➔ This is an optional class provided in MapReduce driver class.

➔ Combiner process the output of map tasks and sends it to the Reducer.

➔ For every mapper, there will be one Combiner.

➔ Combiners are treated as local reducers.

➔ Hadoop does not provide any guarantee on combiner’s execution.

➔ Hadoop may not call combiner function if it is not required.

➔ Hadoop may call one or many times for a map output based on the requirement.

Partitioner: -

➔ Partitioner allows distributing how outputs from the map stage are send to the reducers.

➔ Partitioner controls the keys partition of the intermediate map-outputs.

➔ The key or a subset of the key is used to derive the partition by a hash function.

➔ The total number of partitions is almost same as the number of reduce tasks for the job.

➔ Partitioner runs on the same machine where the mapper had completed its execution by consuming the mapper output.

➔ Entire mapper output sent to partitioner.

➔ Partitioner forms number of reduce task groups from the mapper output.

➔ By default, Hadoop framework is hash based partitioner.

➔ The Hash partitioner partitions the key space by using the hash code.

Shuffling and Sorting: -

➔ The output of the partitioner is Shuffled to the reduce node.

➔ The shuffling is the physical movement of the data over the network.

➔ Once the mappers finished their process, the output produced are shuffled on reducer nodes.

➔ The mapper output is called as intermediate output and it is merged and then sorted.

➔ The sorted output is provided as a input to the reducer phase.

Reducer: -

➔ After the map phase is over, all the intermediate values for the intermediate keys are combined into a list.

➔ Reducer task, which takes the output from a mapper as an input and combines those data tuples into a smaller set of tuples.

➔ There may be single reducer, multiple reducers.

➔ All the values associated with an intermediate key are guaranteed to go to the same reducer.

➔ The intermediate key and their value lists are passed to the reducer in sorted key order.

➔ The reducer outputs zero or more final key/value pairs and these are written to HDFS.

RecordWriter & OutputFormat: -

➔ RecordWriter writes these output key-value pair from the Reducer phase to the output files.

➔ The way of writing the output key-value pairs to output files by RecordWriter is determined by the OutputFormat.

➔ OutputFormat instances provided by the Hadoop are used to write files in HDFS or on the local disk.

➔ The final output of reducer is written on HDFS by OutputFormat instances.

12. Compare R and SAS for data analytics jobs.

SAS	R
A market leader in commercial analytics space	An effective data handling and storage facility
Provides a graphical point-and-click user interface	A comprehensive and integrated collection of intermediate tools for data analysis
Data can be published in HTML, PDF, Excel, and other formats using the Output Delivery System	Graphical facilities for data analysisand display can be done either in soft or hard copy

Can retrieve data from various sources and perform statistical analysis	An effective programming language which includes conditionals, loops, user-defined recursive functions, and input and output facilities
Availability Expensive	Open Source
Improvement Tools Added with New Version	Improvement Tools Quickly Added
Debugging is Easy	Debugging is Difficult
Graphical Capabilities Limited	Graphical Capabilities Advanced
Ease of Learning SAS is very good when it comes to picking a new tool to learn without any prior programming language experience.	R is bit tougher to learn as compared to SAS. Before learning R, you must have a basic knowledge of programming.
Managing Data In terms of handling and managing data, SAS is in a better position since the data is increasing at a huge pace day by day and SAS is better at handling data.	R works only on RAM, and increasing the RAM as and when the data increases is not a feasible option.
Graphics SAS is not great at graphical capabilities. Though Base SAS has some graphical capabilities improvisation, these capabilities are not widely known, and so R gets a clear lead in this aspect.	Graphics is a very important aspect of any Data Science or Data Analytics capabilities. Ability to visualize and analyze data is a crucial part. R is the winner in this area, thanks to the availability of various packages like ggplot, Latice, and RGIS.
Working with Big Data SAS is taking fast strides to execute analytics within Hadoop without the need to move cluster data. But still, SAS lags R when it comes to integrating successfully with Big Data tools like Hadoop and others.	While working with Big Data, R has some very good features which can be utilized by Big Data, Data Science, and Data Analytics communities. R has very good integration with Hadoop, and it also has a parallelization capability. If you are looking for deploying analytics at scale for Machine Learning

	capabilities, then R is the language to choose.

13. What is MADlib? How is it helpful in performing in-database analytics?

MADlib is a free, open-source library of in-database analytic methods.It provides an evolving suite of SQL-based algorithms for machine learning, data mining and statistics that run at scale within a database engine, with no need for data import/export to other tools.

14. Justify that "Hadoop is becoming the corner stone of big data". What aspects of Hadoop are especially relevant to the management and analysis of big data.

Hadoop is the rising star of the business technology agenda for a simple reason — it disrupts the economics of data, analytics, and someday soon, all enterprise applications; it is secretly becoming an application platform too.

➔ **Hadooponomics makes enterprise adoption mandatory:-** Hadoop has been found not guilty of being a hyped-up open source platform. Hadoop has proven real value in any number of use cases including data lakes, traditional and advanced analytics, transactional data, and more. "Hadooponomics" — its ability to linearly scale both data storage and data processing and leverage pay-per-use public cloudonomics

➔ **SQL becomes Hadoop's killer app for 2015**:- SQL is the primary lingua franca for structured enterprise data and is used by application developers to read and write data to databases. It is often used by business intelligence professionals to explore data.SQL on Hadoop creates an instant, easy-to-use and justify use case for enterprises

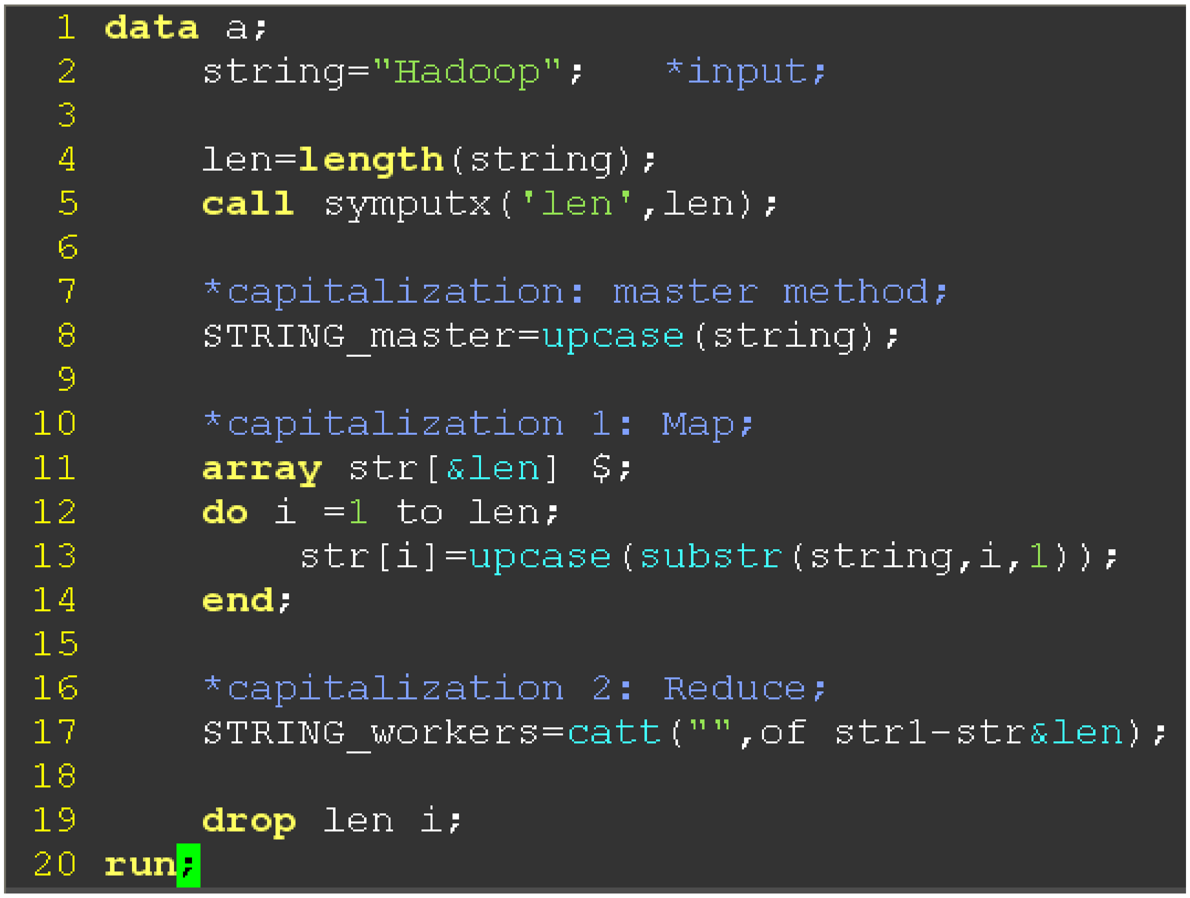
➔ **Enterprise software vendors close Hadoop's data management and governance gaps:-** Hadoop is a general-purpose platform. That means it can store and process any kind or data. The bad news is that all parties agree that Hadoop - based data management and governance solutions have a ways to go before they provide the functionality sophisticated enterprises expect from their app platforms.

➔ **The Hadoop skills shortage disappears**:- Hadoop is not that hard to understand. It is a file system, albeit distributed, and it is a computing platform, albeit distributed. The APIs are Java. CEO's won't have to hire high-priced Hadoop consultants to get projects done.

➔ **Enterprises will let thousands of Hadoop clusters bloom in the cloud**:- Hadoop is both a data storage and data processing system. That means storage, compute, and network resources are required to run a Hadoop cluster. Early adopters of Hadoop will increasingly use Hadoop in the cloud to optimize the cost of Hadoop clusters and to meet demand for ad hoc analytics on Hadoop

➔ **Hadoop won't be just for analytics anymore:-** Hadoop is as much a computing platform that can become the foundational component of enterprise applications. With better resource management features provided by YARN, database options such as HBase, and in-memory overlay Apache Spark, Hadoop becomes an application platform.

15. How do you perform map reduce in SAS.



```
 1 data a;
 2      string="Hadoop";    *input;
 3
 4      len=length(string);
 5      call symputx('len',len);
 6
 7      *capitalization: master method;
 8      STRING_master=upcase(string);
 9
10      *capitalization 1: Map;
11      array str[&len] $;
12      do i =1 to len;
13          str[i]=upcase(substr(string,i,1));
14      end;
15
16      *capitalization 2: Reduce;
17      STRING_workers=catt("",of str1-str&len);
18
19      drop len i;
20 run;
```

the programming task is to capitalize a string "Hadoop" (Line 2) and the "master" method is just to capitalize the string in buddle(Line 8): just use a master machine to processing the data.

Then we introduce the idea of "big data" that the string is too huge to one master machine, so "master method" failed. Now we distribute the task to thousands of low cost machines (workers, slaves, chunk servers,… in this case, the one dimensional array with size of 6, see Line 11), each machine produces parts of the job (each array element only capitalizes a single letter in sequence, see Line 12-14). Such distributing operation is called "map". In a MapReduce system, a master machine is also needed to assign the maps and reduce.

How about "reduce"? A "reduce" operation is also called "fold"—for example, in Line 17, the operation to combine all the separately values into a single value: combine results from multiple worker machines.