RAISONI GROUP
a vision beyond

G. H. Raisoni College of Engineering & Management, Wagholi, Pune – 412 207

Department of Computer Engineering

D-19

Lab Manual (2021-22)

Pattern-2016

Class: B. Tech.

Data Science and Big Data Lab (BCOP404A)

Faculty Name: Sarita Patil

G. H. Raisoni College of Engineering and Management, Wagholi, Pune

Department: Computer Engineering

Course Details

Course : Data Science & Big Data Lab (BCOP404A)

Class: B. Tech.	**Division: A and B**
Internal Marks: 25	**External Marks: 25**
Credits: 1	**Pattern: 2016**

COURSE OUTCOMES	
CO1	To understand the basics of big data and Hadoop.
CO2	To Learn SQL/NoSQL in data science.
CO3	To understand the data formats of numpy and pandas
CO4	To understand the process of data preprocessing
CO5	To perform statistical data analysis and visualize the data using libraries.

List of Experiments

Sr. No.	Experiment List	CO Mapping	Software Required
1	Big data, map, reduce and HDFS operations.	CO1	Fedora
2	Hive installation, configuration and implementation.	CO1	Fedora
3	SQL/NoSQL programming for data science.	CO1, CO2	Anaconda, SQL
4	Understanding data formats of Numpy: Numpy array operations, Slicing and exercise	CO3	Anaconda, SQL
5	Understanding data formats of Pandas: Series, Data Frame, Handling missing data, grouping data, Read/write CSV, Excel, HTML, JSON data	CO3	Anaconda, SQL
6	Data Visualization: Basic plotting, subplots, Histogram,box plot, bar chart, pie chart‘ line chart	CO3,CO5	Anaconda, SQL
7	Data Preprocessing/Wrangling: Basic data handling, reshaping, pivoting, rank and sort data, basic data grouping	CO3,CO4	Anaconda, SQL
8	Data Preprocessing/Wrangling: ETL Phase 1 and Phase 2	CO3,CO4	Anaconda, SQL
9	Web Scrapping	CO3,CO4	Anaconda, SQL
10	Data transformation and Exploratory Data Analysis	CO3,CO4, CO5	Anaconda, SQL
	Content Beyond Syllabus		
1	Perform the following operations using R on the sample data set a)Create data subsets b)Merge data c)Sort data d) Transposing data e)Melting Data to long format d)Casting Data to wide format	CO4	Anaconda, SQL, R Studio
2	Take Sample dataset and visualize the data using R by plotting different graph.	CO5,CO2	Anaconda, SQL, R Studio

ASSIGNMENT NUMBER 1

TITLE:

Big data, map reduce and HDFS operations

THEORY:

Big data is a collection of large datasets that cannot be processed using traditional computing techniques. It is not a single technique or a tool, rather it has become a complete subject, which involves various tools, techniques and frameworks.

Big data involves the data produced by different devices and applications. Given below are some of the fields that come under the umbrella of Big Data.

- **Black Box Data** − It is a component of helicopter, airplanes, and jets, etc. It captures voices of the flight crew, recordings of microphones and earphones, and the performance information of the aircraft.
- **Social Media Data** − Social media such as Facebook and Twitter hold information and the views posted by millions of people across the globe.
- **Stock Exchange Data** − The stock exchange data holds information about the _buy‘ and _sell‘ decisions made on a share of different companies made by the customers.
- **Power Grid Data** − The power grid data holds information consumed by a particular node with respect to a base station.
- **Transport Data** − Transport data includes model, capacity, distance and availability of a vehicle.
- **Search Engine Data** − Search engines retrieve lots of data from different databases.

Thus Big Data includes huge volume, high velocity, and extensible variety of data. The data in it will be of three types.

- **Structured data** − Relational data.
- **Semi Structured data** − XML data.
- **Unstructured data** − Word, PDF, Text, Media Logs.
- Benefits of Big Data
- Using the information kept in the social network like Facebook, the marketing agencies are learning about the response for their campaigns, promotions, and other advertising mediums.
- Using the information in the social media like preferences and product perception of their consumers, product companies and retail organizations are planning their production.
- Using the data regarding the previous medical history of patients, hospitals are providing better and quick service.

Big Data Technologies

Big data technologies are important in providing more accurate analysis, which may lead to more concrete decision-making resulting in greater operational efficiencies, cost reductions, and reduced risks for the business.

To harness the power of big data, you would require an infrastructure that can manage and process huge volumes of structured and unstructured data in realtime and can protect data privacy and security.

There are various technologies in the market from different vendors including Amazon, IBM, Microsoft, etc., to handle big data. While looking into the technologies that handle big data, we examine the following two classes of technology −

Operational Big Data

This include systems like MongoDB that provide operational capabilities for real-time, interactive workloads where data is primarily captured and stored.

NoSQL Big Data systems are designed to take advantage of new cloud computing architectures that have emerged over the past decade to allow massive computations to be run inexpensively and efficiently. This makes operational big data workloads much easier to manage, cheaper, and faster to implement.

Some NoSQL systems can provide insights into patterns and trends based on real-time data with minimal coding and without the need for data scientists and additional infrastructure.

Analytical Big Data

These includes systems like Massively Parallel Processing (MPP) database systems and MapReduce that provide analytical capabilities for retrospective and complex analysis that may touch most or all of the data.

MapReduce provides a new method of analyzing data that is complementary to the capabilities provided by SQL, and a system based on MapReduce that can be scaled up from single servers to thousands of high and low end machines.

Hadoop

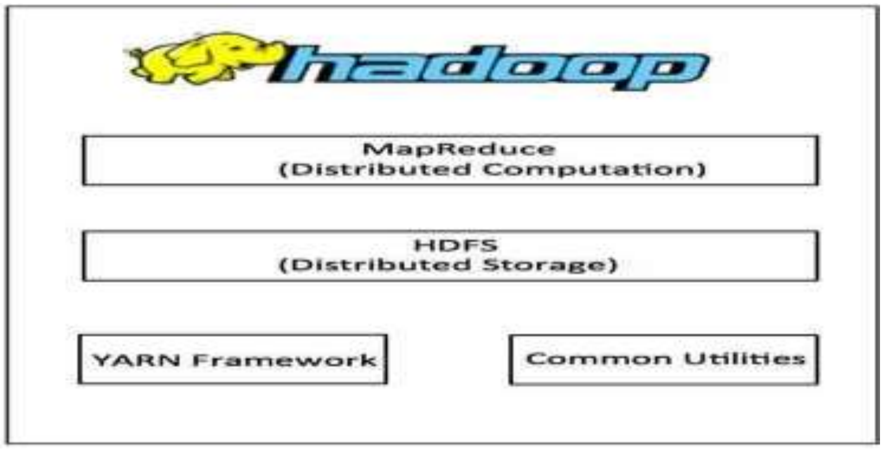
Hadoop is an Apache open source framework written in java that allows distributed processing of large datasets across clusters of computers using simple programming models. The Hadoop framework application works in an environment that provides distributed *storage* and *computation* across clusters of computers. Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Hadoop Architecture

At its core, Hadoop has two major layers namely −

- Processing/Computation layer (MapReduce), and

- Storage layer (Hadoop Distributed File System).



MapReduce

MapReduce is a parallel programming model for writing distributed applications devised at Google for efficient processing of large amounts of data (multi-terabyte data-sets), on large clusters (thousands of nodes) of commodity hardware in a reliable, fault-tolerant manner. The MapReduce program runs on Hadoop which is an Apache open-source framework.

Hadoop Distributed File System

The Hadoop Distributed File System (HDFS) is based on the Google File System (GFS) and provides a distributed file system that is designed to run on commodity hardware. It has many similarities with existing distributed file systems. However, the differences from other distributed file systems are significant. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. It provides high throughput access to application data and is suitable for applications having large datasets.

Apart from the above-mentioned two core components, Hadoop framework also includes the following two modules

- **Hadoop Common** − These are Java libraries and utilities required by other Hadoop modules.
- **Hadoop YARN** − This is a framework for job scheduling and cluster resource management.

How Does Hadoop Work?

It is quite expensive to build bigger servers with heavy configurations that handle large scale processing, but as an alternative, you can tie together many commodity computers with single-CPU, as a single functional distributed system and practically, the clustered machines can read the dataset in parallel and provide a much higher throughput. Moreover, it is cheaper than one high-end server. So this is the first motivational factor behind using Hadoop that it runs across clustered and low-cost machines.

Hadoop runs code across a cluster of computers. This process includes the following core tasks that Hadoop performs −

- Data is initially divided into directories and files. Files are divided into uniform sized blocks of 128M and 64M (preferably 128M).
- These files are then distributed across various cluster nodes for further processing.

- HDFS, being on top of the local file system, supervises the processing.
- Blocks are replicated for handling hardware failure.
- Checking that the code was executed successfully.
- Performing the sort that takes place between the map and reduce stages.
- Sending the sorted data to a certain computer.
- Writing the debugging logs for each job. Advantages of Hadoop
- Hadoop framework allows the user to quickly write and test distributed systems. It is efficient, and it
automatic distributes the data and work across the machines and in turn, utilizes the underlying parallelism of the CPU cores.

- Hadoop does not rely on hardware to provide fault-tolerance and high availability (FTHA), rather Hadoop library itself has been designed to detect and handle failures at the application layer.
- Servers can be added or removed from the cluster dynamically and Hadoop continues to operate without interruption.
- Another big advantage of Hadoop is that apart from being open source, it is compatible on all the platforms since it is Java based.

HDFS

Hadoop File System was developed using distributed file system design. It is run on commodity hardware. Unlike other distributed systems, HDFS is highly faulttolerant and designed using low-cost hardware.

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 Architecture

Given below is the architecture of a Hadoop File System.

HDFS follows the master-slave architecture and it has the following elements.

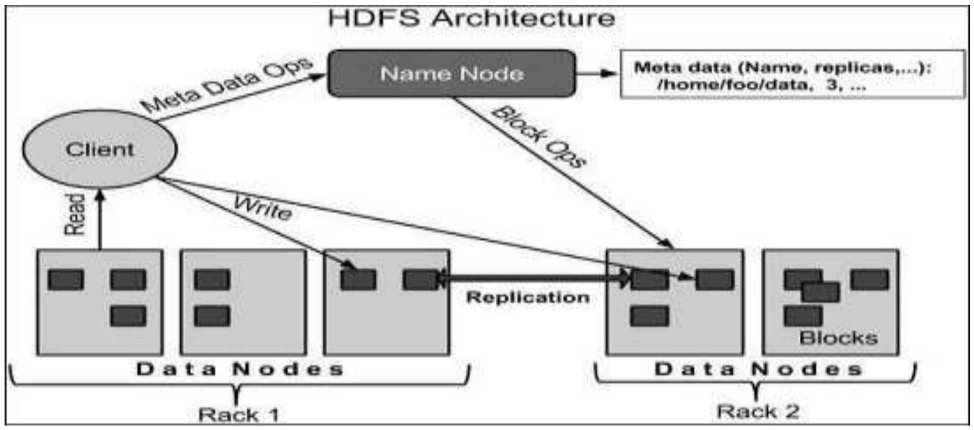


Fig No. 1 Hadoop Architecture

Name node

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

- Manages the file system namespace.
- Regulates clients’ access to files.
- It also executes file system operations such as renaming, closing, and opening files and directories.

Data node

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

- Data nodes 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 name node.

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. In other words, the minimum amount of data that HDFS can read or write is called a Block. The default block size is 64MB, but it can be increased as per the need to change in HDFS configuration.

Goals of HDFS

Fault detection and recovery − Since HDFS includes a large number of commodity hardware, failure of components is frequent. Therefore HDFS should have mechanisms for quick and automatic fault detection and recovery.

Huge datasets − HDFS should have hundreds of nodes per cluster to manage the applications having huge datasets.

Hardware at data − A requested task can be done efficiently, when the computation takes place near the data. Especially where huge datasets are involved, it reduces the network traffic and increases the throughput.

EXERCISE :

A) Installation of Hadoop On Ubantu:

Prerequisites

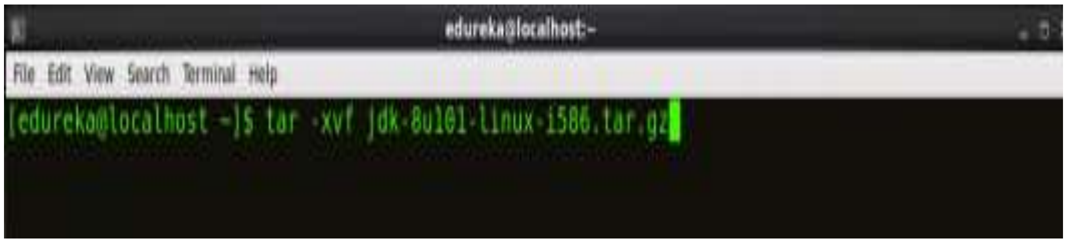
- *VIRTUAL BOX*: it is used for installing the operating system on it.
- *OPERATING SYSTEM*: You can install Hadoop on Linux based operating systems. Ubuntu and CentOS are very commonly used. In this tutorial, we are using CentOS.
- *JAVA*: You need to install the Java 8 package on your system.
- *HADOOP*: You require Hadoop 2.7.3 package.

Install Hadoop

Step 1: To download the Java 8 Package. Save this file in your home directory.

Step 2: Extract the Java Tar File.

***Command*:** tar -xvf jdk-8u101-linux-i586.tar.gz

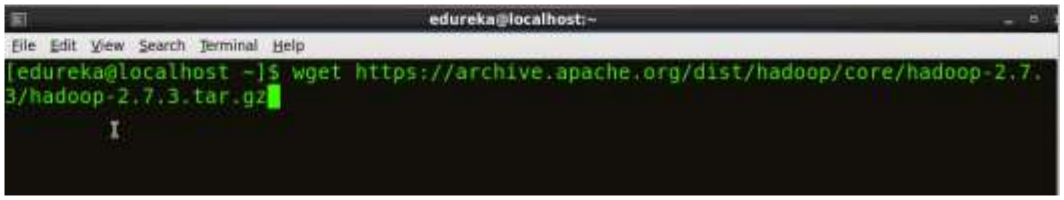


```
[edureka@localhost ~]$ tar -xvf jdk-8u101-linux-i586.tar.gz
```

Fig 2: Hadoop Installation – Extracting Java Files

Step 3: Download the Hadoop 2.7.3 Package.

***Command*:** wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.3/hadoop-2.7.3.tar.gz

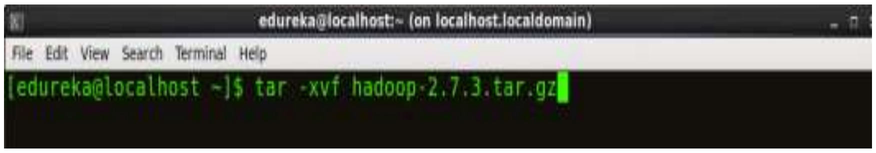


```
[edureka@localhost ~]$ wget https://archive.apache.org/dist/hadoop/core/hadoop-2.7.
3/hadoop-2.7.3.tar.gz
```

Fig 3: Hadoop Installation – Downloading Hadoop

Step 4: Extract the Hadoop tar File.

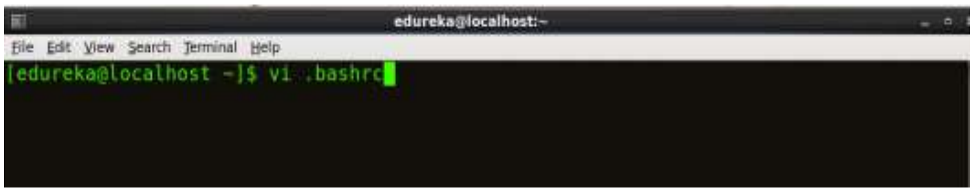
Command: tar -xvf hadoop-2.7.3.tar.gz



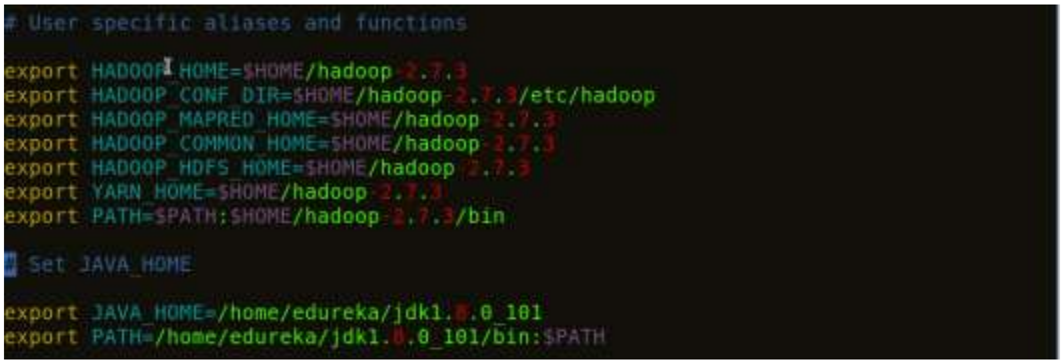
```
[edureka@localhost ~]$ tar -xvf hadoop-2.7.3.tar.gz
```

Fig4: Hadoop Installation – Extracting Hadoop Files

Step 5: Add the Hadoop and Java paths in the bash file (.bashrc). Open**. bashrc** file. Now, add Hadoop and Java Path as shown below. ***Command*:** vi .bashrc



```
[edureka@localhost ~]$ vi .bashrc
```



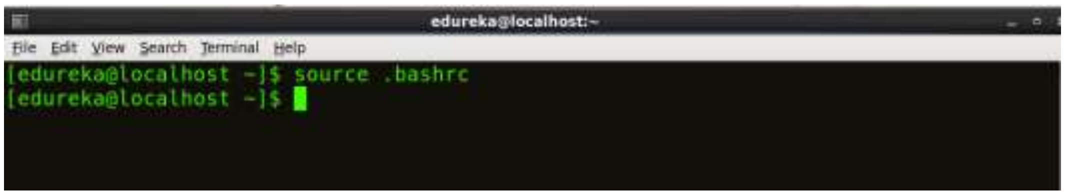
```
# User specific aliases and functions

export HADOOP_HOME=$HOME/hadoop-2.7.3
export HADOOP_CONF_DIR=$HOME/hadoop-2.7.3/etc/hadoop
export HADOOP_MAPRED_HOME=$HOME/hadoop-2.7.3
export HADOOP_COMMON_HOME=$HOME/hadoop-2.7.3
export HADOOP_HDFS_HOME=$HOME/hadoop-2.7.3
export YARN_HOME=$HOME/hadoop-2.7.3
export PATH=$PATH:$HOME/hadoop-2.7.3/bin

# Set JAVA_HOME

export JAVA_HOME=/home/edureka/jdk1.8.0_101
export PATH=/home/edureka/jdk1.8.0_101/bin:$PATH
```

Fig5: Hadoop Installation – Setting Environment Variable

Then, save the bash file and close it.

For applying all these changes to the current Terminal, execute the source command.

***Command*:** source .bashrc

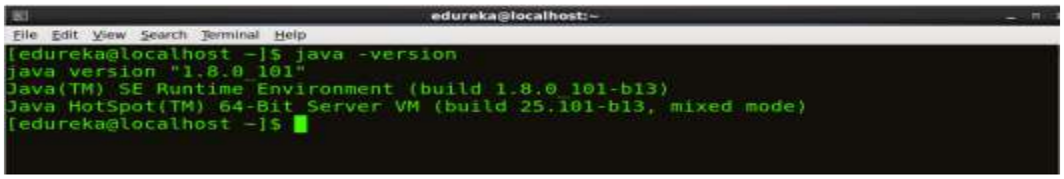


```
[edureka@localhost ~]$ source .bashrc
[edureka@localhost ~]$
```

Fig 6 : Hadoop Installation – Refreshing environment variables

To make sure that Java and Hadoop have been properly installed on your system and can be accessed through the Terminal, execute the java -version and hadoop version commands.

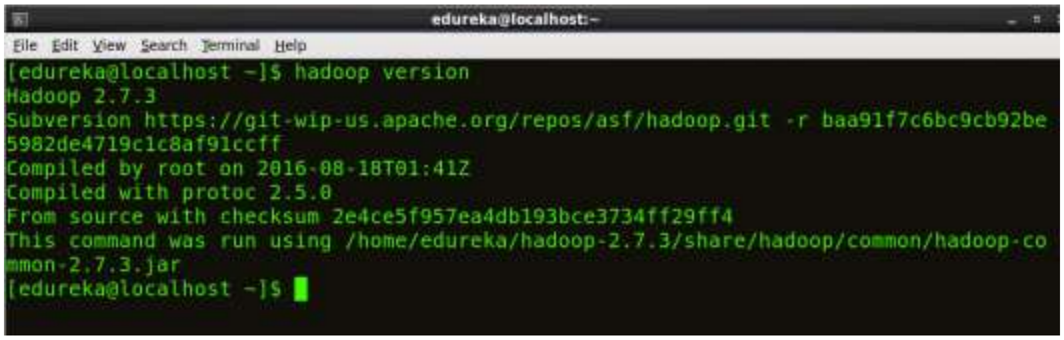
***Command*:** java -version



```
[edureka@localhost ~]$ java -version
java version "1.8.0_101"
Java(TM) SE Runtime Environment (build 1.8.0_101-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)
[edureka@localhost ~]$
```

Fig 7: Hadoop Installation – Checking Java Version

***Command*:** hadoop version

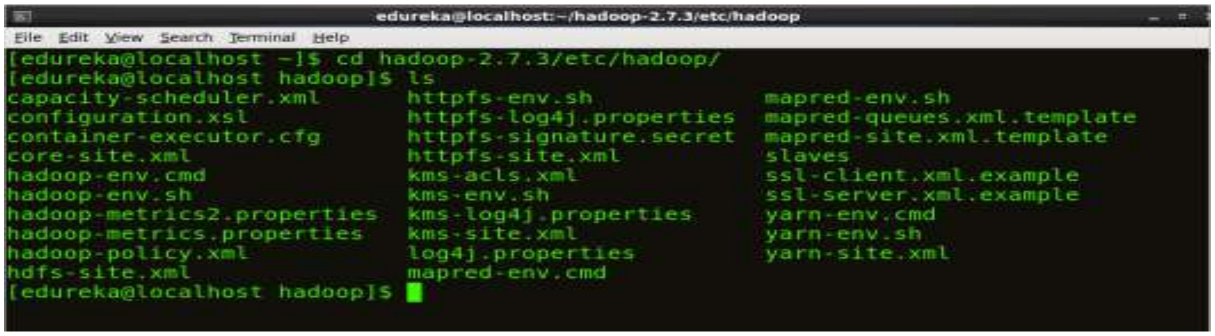


```
[edureka@localhost ~]$ hadoop version
Hadoop 2.7.3
Subversion https://git-wip-us.apache.org/repos/asf/hadoop.git -r baa91f7c6bc9cb92be
5982de4719c1c8af91ccff
Compiled by root on 2016-08-18T01:41Z
Compiled with protoc 2.5.0
From source with checksum 2e4ce5f957ea4db193bce3734ff29ff4
This command was run using /home/edureka/hadoop-2.7.3/share/hadoop/common/hadoop-co
mmon-2.7.3.jar
[edureka@localhost ~]$
```

Fig 8: Hadoop Installation – Checking Hadoop Version

Step 6: Edit the **Hadoop Configuration files**. ***Command:*** cd hadoop-2.7.3/etc/hadoop/ ***Command:*** ls

the Hadoop configuration files are located in **hadoop-2.7.3/etc/hadoop** directory as you can see in the snapshot below:



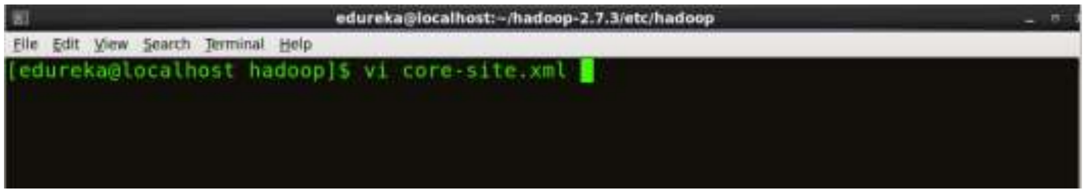
```
[edureka@localhost ~]$ cd hadoop-2.7.3/etc/hadoop/
[edureka@localhost hadoop]$ ls
capacity-scheduler.xml      httpfs-env.sh             mapred-env.sh
configuration.xsl           httpfs-log4j.properties   mapred-queues.xml.template
container-executor.cfg      httpfs-signature.secret   mapred-site.xml.template
core-site.xml               httpfs-site.xml           slaves
hadoop-env.cmd              kms-acls.xml              ssl-client.xml.example
hadoop-env.sh               kms-env.sh                ssl-server.xml.example
hadoop-metrics2.properties  kms-log4j.properties      yarn-env.cmd
hadoop-metrics.properties   kms-site.xml              yarn-env.sh
hadoop-policy.xml           log4j.properties          yarn-site.xml
hdfs-site.xml               mapred-env.cmd
[edureka@localhost hadoop]$
```

Fig 9: Hadoop Installation – Hadoop Configuration Files

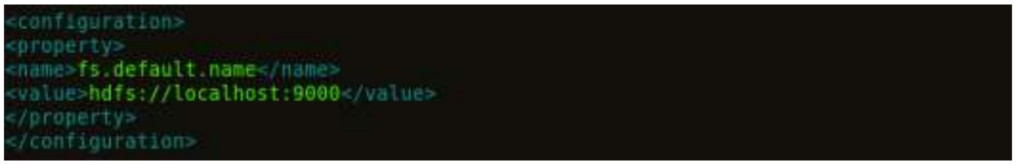
Step 7: Open *core-site.xml* and edit the property mentioned below inside configuration tag:

core-site.xml informs Hadoop daemon where NameNode runs in the cluster. It contains configuration settings of Hadoop core such as I/O settings that are common to HDFS & MapReduce.

***Command*:** vi core-site.xml



```
[edureka@localhost hadoop]$ vi core-site.xml
```



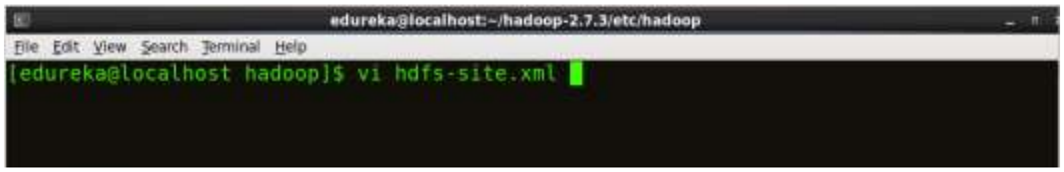
```
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

Fig10: Hadoop Installation – Configuring core-site.xml

Step 8: Edit *hdfs-site.xml* and edit the property mentioned below inside configuration tag:

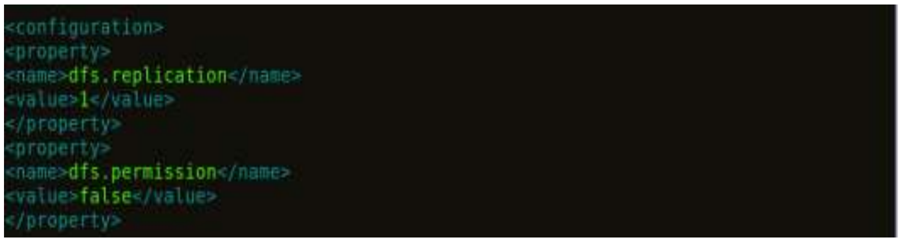
hdfs-site.xml contains configuration settings of HDFS daemons (i.e. NameNode, DataNode, Secondary NameNode). It also includes the replication factor and block size of HDFS.

Command**:** vi hdfs-site.xml



edureka@localhost:~/hadoop-2.7.3/etc/hadoop
File Edit View Search Terminal Help

```
[edureka@localhost hadoop]$ vi hdfs-site.xml
```



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

Fig11: Hadoop Installation – Configuring hdfs-site.xml

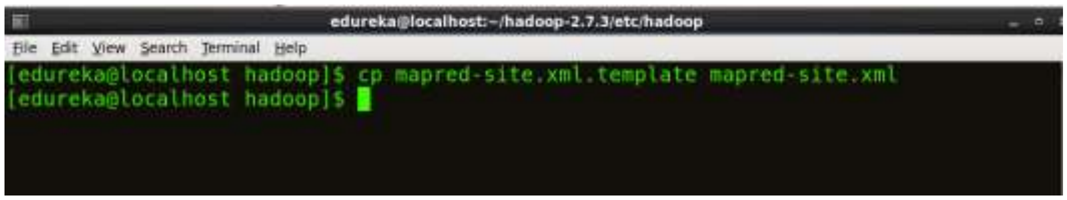
Step 9: Edit the *mapred-site.xml* file and edit the property mentioned below inside configuration tag:

mapred-site.xml contains configuration settings of MapReduce application like number of JVM that can run in parallel, the size of the mapper and the reducer process, CPU cores available for a process, etc.

In some cases, mapred-site.xml file is not available. So, we have to create the mapred-site.xml file using mapred-site.xml template.

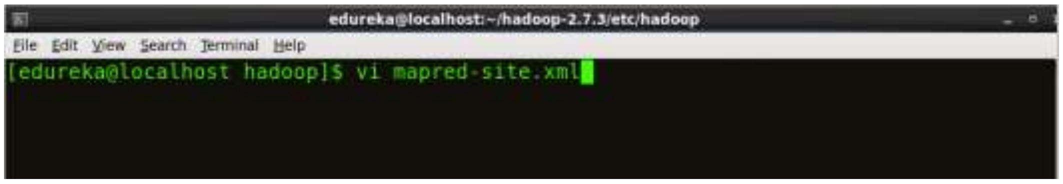
Command**:** cp mapred-site.xml.template mapred-site.xml

Command**:** vi mapred-site.xml.

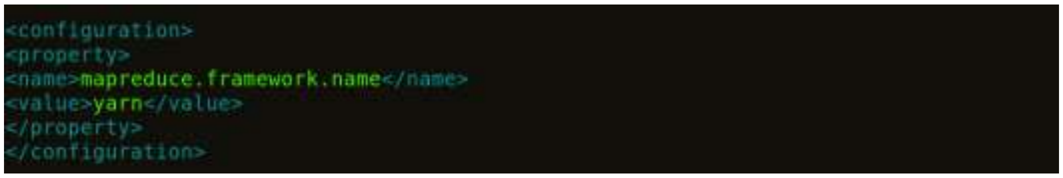


edureka@localhost:~/hadoop-2.7.3/etc/hadoop
File Edit View Search Terminal Help

```
[edureka@localhost hadoop]$ cp mapred-site.xml.template mapred-site.xml
[edureka@localhost hadoop]$ 
```



```
[edureka@localhost hadoop]$ vi mapred-site.xml
```



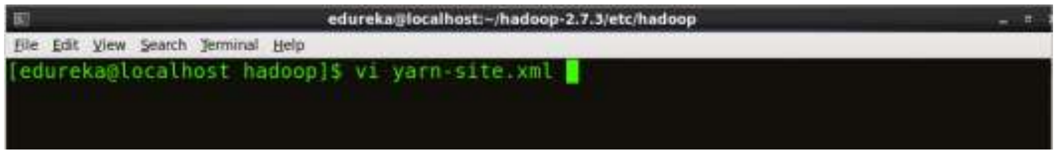
```
<configuration>
<property>
<name>mapreduce.framework.name</name>
<value>yarn</value>
</property>
</configuration>
```

Fig 12: Hadoop Installation – Configuring mapred-site.xml

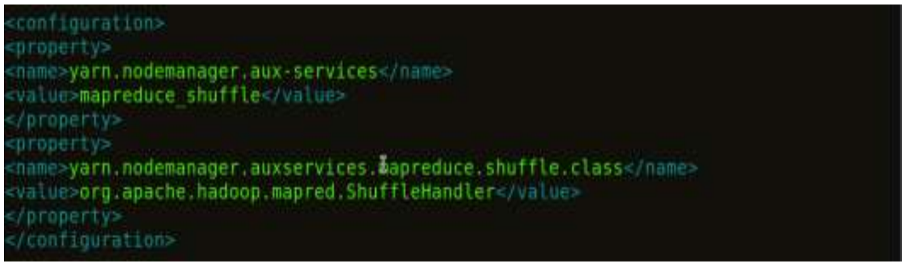
Step 10: Edit *yarn-site.xml* and edit the property mentioned below inside configuration tag:

yarn-site.xml contains configuration settings of ResourceManager and NodeManager like application memory management size, the operation needed on program & algorithm, etc.

***Command*:** vi yarn-site.xml



```
[edureka@localhost hadoop]$ vi yarn-site.xml
```



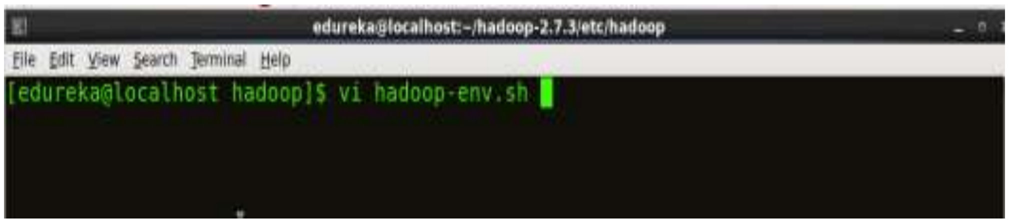
```
<configuration>
<property>
<name>yarn.nodemanager.aux-services</name>
<value>mapreduce_shuffle</value>
</property>
<property>
<name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
<value>org.apache.hadoop.mapred.ShuffleHandler</value>
</property>
</configuration>
```

Fig 13: Hadoop Installation – Configuring yarn-site.xml

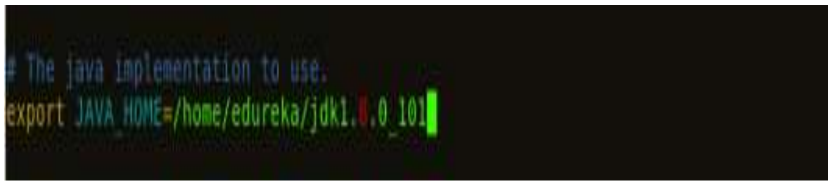
Step 11: Edit *hadoop-env.sh* and add the Java Path as mentioned below:

hadoop-env.sh contains the environment variables that are used in the script to run Hadoop like Java home path, etc.

***Command*:** vi hadoop–env.sh



```
[edureka@localhost hadoop]$ vi hadoop-env.sh
```



```
# The java implementation to use.
export JAVA_HOME=/home/edureka/jdk1.8.0_101
```

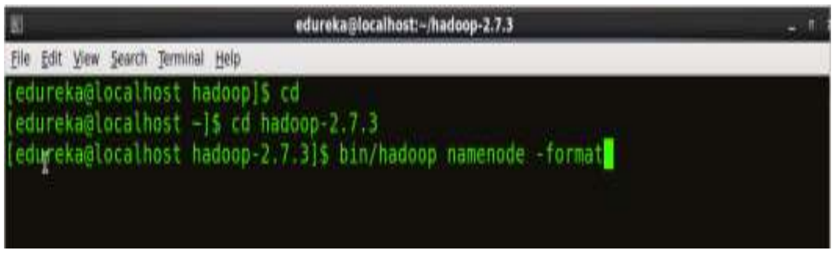
Fig 14: Hadoop Installation – Configuring hadoop-env.sh

Step 12: Go to Hadoop home directory and format the NameNode.

Command: cd

Command: cd hadoop-2.7.3

Command: bin/hadoop namenode -format



```
[edureka@localhost hadoop]$ cd
[edureka@localhost ~]$ cd hadoop-2.7.3
[edureka@localhost hadoop-2.7.3]$ bin/hadoop namenode -format
```

Fig 15: Hadoop Installation – Formatting NameNode

This formats the HDFS via NameNode. This command is only executed for the first time. Formatting the file system means initializing the directory specified by the dfs.name.dir variable.

Never format, up and running Hadoop filesystem. You will lose all your data stored in the HDFS.

Step 13: Once the NameNode is formatted, go to hadoop-2.7.3/sbin directory and start all the daemons.

Command: cd hadoop-2.7.3/sbin

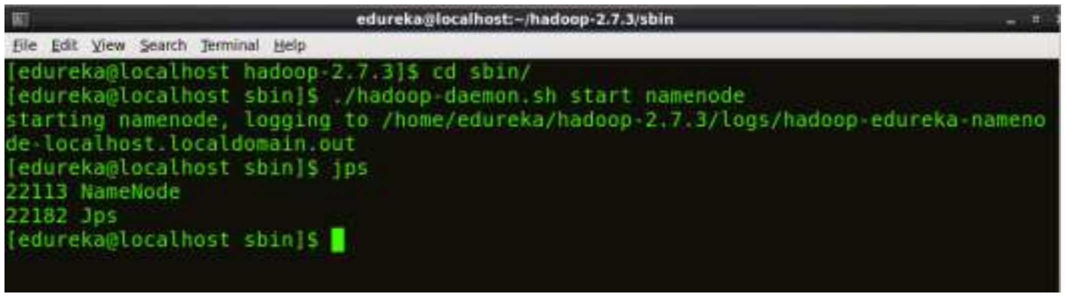
Either you can start all daemons with a single command or do it individually.

Command: ./start-all.sh

The above command is a combination of ***start-dfs.sh, start-yarn.sh*** & ***mr-jobhistory-daemon.sh***

Or you can run all the services individually as below:

start NameNode: The NameNode is the centerpiece of an HDFS file system. It keeps the directory tree of all files stored in the HDFS and tracks all the file stored across the cluster.



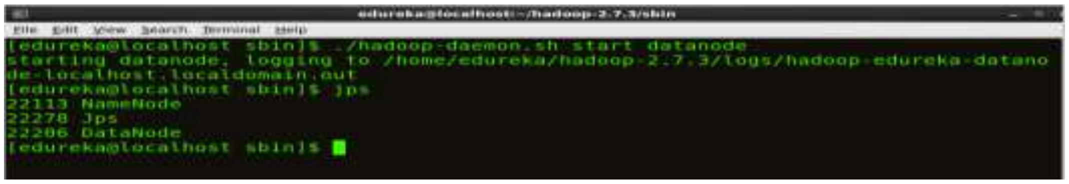
```
[edureka@localhost hadoop-2.7.3]$ cd sbin/
[edureka@localhost sbin]$ ./hadoop-daemon.sh start namenode
starting namenode, logging to /home/edureka/hadoop-2.7.3/logs/hadoop-edureka-nameno
de-localhost.localdomain.out
[edureka@localhost sbin]$ jps
22113 NameNode
22182 Jps
[edureka@localhost sbin]$
```

Command: ./hadoop-daemon.sh start namenode

Fig16: Hadoop Installation – Starting NameNode

Start DataNode:

On startup, a DataNode connects to the Namenode and it responds to the requests from the Namenode for different operations.



```
[edureka@localhost sbin]$ ./hadoop-daemon.sh start datanode
starting datanode, logging to /home/edureka/hadoop-2.7.3/logs/hadoop-edureka-datano
de-localhost.localdomain.out
[edureka@localhost sbin]$ jps
22113 NameNode
22278 Jps
22206 DataNode
[edureka@localhost sbin]$
```

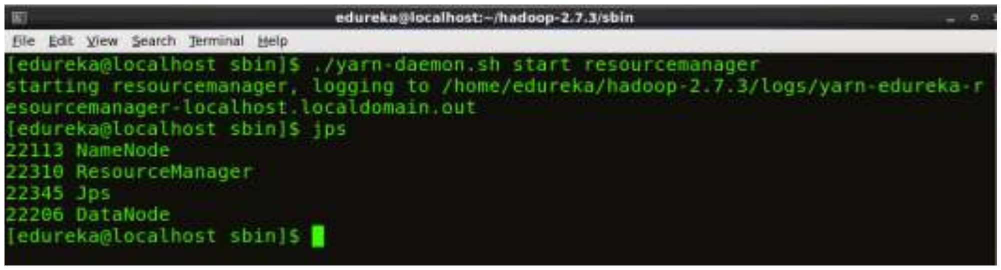
Fig17: Hadoop Installation – Starting DataNode

Command: ./hadoop-daemon.sh start datanode

Start ResourceManager:

ResourceManager is the master that arbitrates all the available cluster resources and thus helps in managing the distributed applications running on the YARN system. Its work is to manage each NodeManagers and the each application's ApplicationMaster.

Command: ./yarn-daemon.sh start resourcemanager



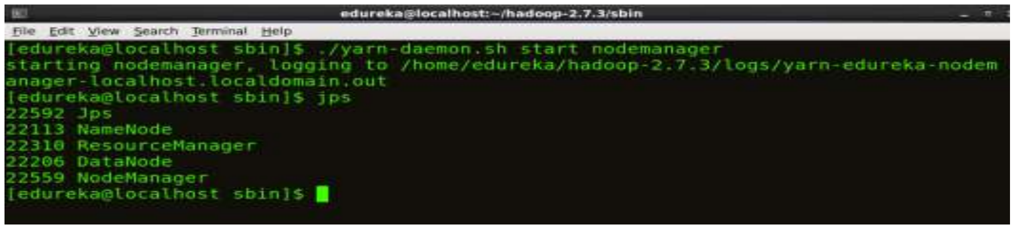
```
[edureka@localhost sbin]$ ./yarn-daemon.sh start resourcemanager
starting resourcemanager, logging to /home/edureka/hadoop-2.7.3/logs/yarn-edureka-r
esourcemanager-localhost.localdomain.out
[edureka@localhost sbin]$ jps
22113 NameNode
22310 ResourceManager
22345 Jps
22206 DataNode
[edureka@localhost sbin]$
```

Fig 18: Hadoop Installation – Starting ResourceManager

Start Node Manager:

The NodeManager in each machine framework is the agent which is responsible for managing containers, monitoring their resource usage and reporting the same to the ResourceManager.

Command: ./yarn-daemon.sh start nodemanager



```
[edureka@localhost sbin]$ ./yarn-daemon.sh start nodemanager
starting nodemanager, logging to /home/edureka/hadoop-2.7.3/logs/yarn-edureka-nodem
anager-localhost.localdomain.out
[edureka@localhost sbin]$ jps
22592 Jps
22113 NameNode
22310 ResourceManager
22206 DataNode
22559 NodeManager
[edureka@localhost sbin]$
```

Fig 19: Hadoop Installation – Starting NodeManager

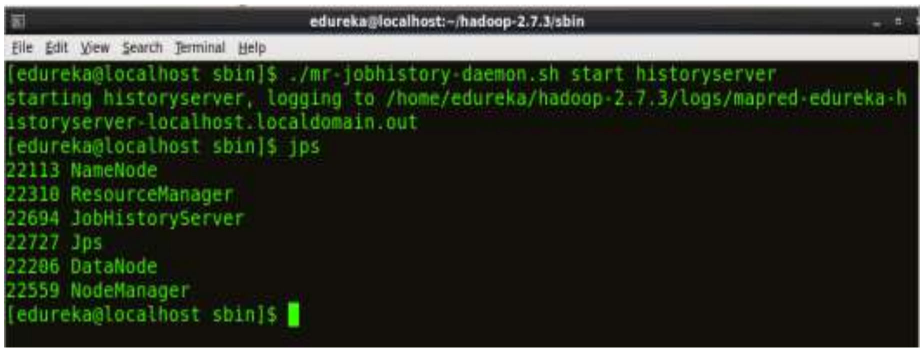
Start Job History Server:

Job History Server is responsible for servicing all job history related requests from client.

Command: ./mr-jobhistory-daemon.sh start historyserver

Step 14: To check that all the Hadoop services are up and running, run the below command.

Command: jps



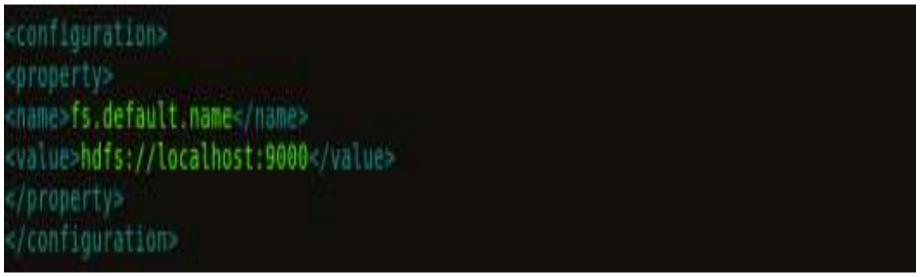
edureka@localhost:~/hadoop-2.7.3/sbin
File Edit View Search Terminal Help

```
[edureka@localhost sbin]$ ./mr-jobhistory-daemon.sh start historyserver
starting historyserver, logging to /home/edureka/hadoop-2.7.3/logs/mapred-edureka-h
istoryserver-localhost.localdomain.out
[edureka@localhost sbin]$ jps
22113 NameNode
22310 ResourceManager
22694 JobHistoryServer
22727 Jps
22206 DataNode
22559 NodeManager
[edureka@localhost sbin]$ 
```

Fig 20: Hadoop Installation – Checking Daemons

Step 15: Now open the Mozilla browser and go to **localhost**:**50070/dfshealth.html** to check the NameNode interface.

All the Hadoop configuration files are located in **hadoop-2.7.3/etc/hadoop** directory as you can see in the snapshot below:



```
<configuration>
<property>
<name>fs.default.name</name>
<value>hdfs://localhost:9000</value>
</property>
</configuration>
```

Fig 21: Hadoop Installation – Configuring core-site.xml

B) <u>Step by step Hadoop 2.8.0 installation on Window 10</u>

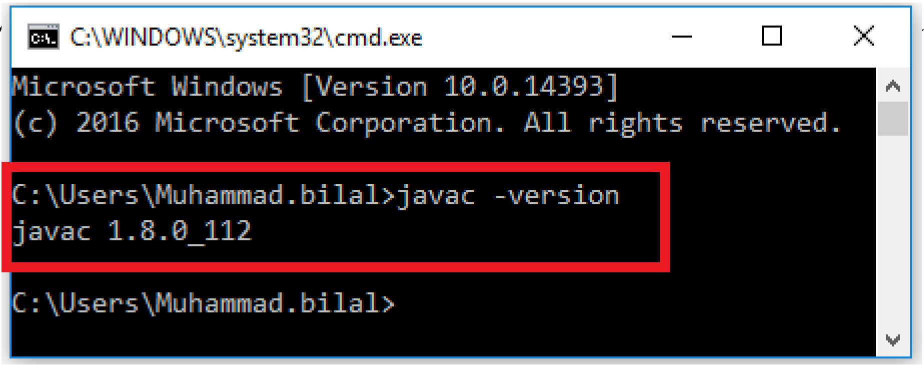
3. Prepare:

These softwares should be prepared to install Hadoop 2.8.0 on window 10 64bit

1. Download Hadoop 2.8.0 (<u>Link 1</u> OR <u>Link 2</u>)
2. Java JDK 1.8.0.zip <u>Link to download</u>

4. Set up

1. Check either Jav

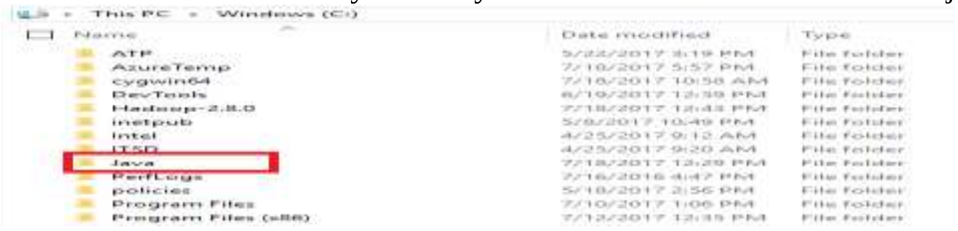


C:\WINDOWS\system32\cmd.exe

```
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Muhammad.bilal>javac -version
javac 1.8.0_112

C:\Users\Muhammad.bilal>
```

vac -
version” tocheck.

2. If Java is not installed on your system then first install java

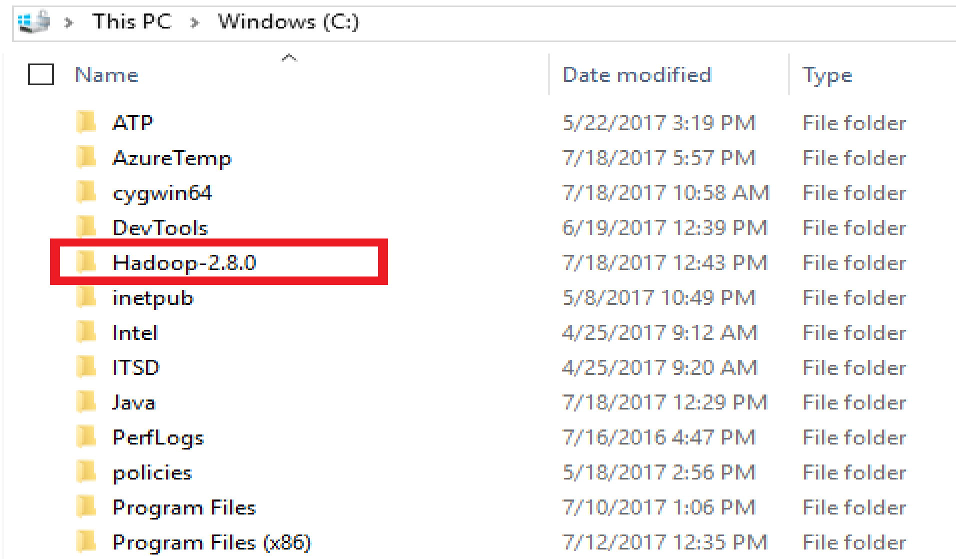


This PC › Windows (C:)

Name	Date modified	Type
ATP	5/22/2017 3:19 PM	File folder
AzureTemp	7/18/2017 5:57 PM	File folder
cygwin64	7/18/2017 10:58 AM	File folder
DevTools	6/19/2017 12:39 PM	File folder
Hadoop-2.8.0	7/18/2017 12:43 PM	File folder
inetpub	5/8/2017 10:49 PM	File folder
Intel	4/25/2017 9:12 AM	File folder
ITSD	4/25/2017 9:20 AM	File folder
Java	7/18/2017 12:29 PM	File folder
PerfLogs	7/16/2016 4:47 PM	File folder
policies	5/18/2017 2:56 PM	File folder
Program Files	7/10/2017 1:06 PM	File folder
Program Files (x86)	7/12/2017 12:35 PM	File folder

under **“C:\JAVA”**

3. Extract file Hadoop 2.8.0.tar.gz or Hadoop-2.8.0.zip and place under **“C:\Hadoop2.8.0”**.

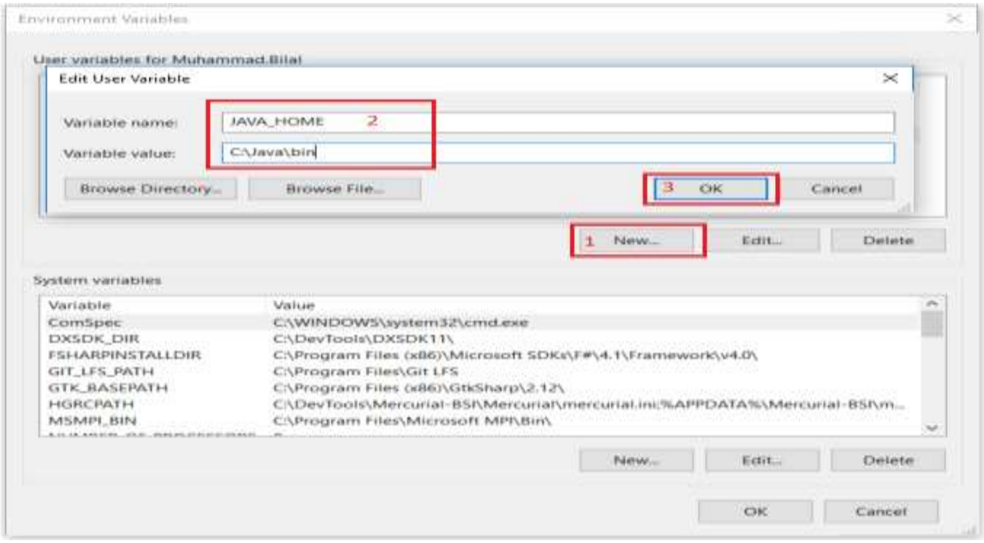


This PC › Windows (C:)

Name	Date modified	Type
ATP	5/22/2017 3:19 PM	File folder
AzureTemp	7/18/2017 5:57 PM	File folder
cygwin64	7/18/2017 10:58 AM	File folder
DevTools	6/19/2017 12:39 PM	File folder
Hadoop-2.8.0	7/18/2017 12:43 PM	File folder
inetpub	5/8/2017 10:49 PM	File folder
Intel	4/25/2017 9:12 AM	File folder
ITSD	4/25/2017 9:20 AM	File folder
Java	7/18/2017 12:29 PM	File folder
PerfLogs	7/16/2016 4:47 PM	File folder
policies	5/18/2017 2:56 PM	File folder
Program Files	7/10/2017 1:06 PM	File folder
Program Files (x86)	7/12/2017 12:35 PM	File folder

4. Set the path HADOOP_HOME Environment variable on windows 10(see Step 1,2,3 and 4 below).

5. Set the path JAVA_HOME Environment variable on windows 10(see Step 1,2,3 and 4

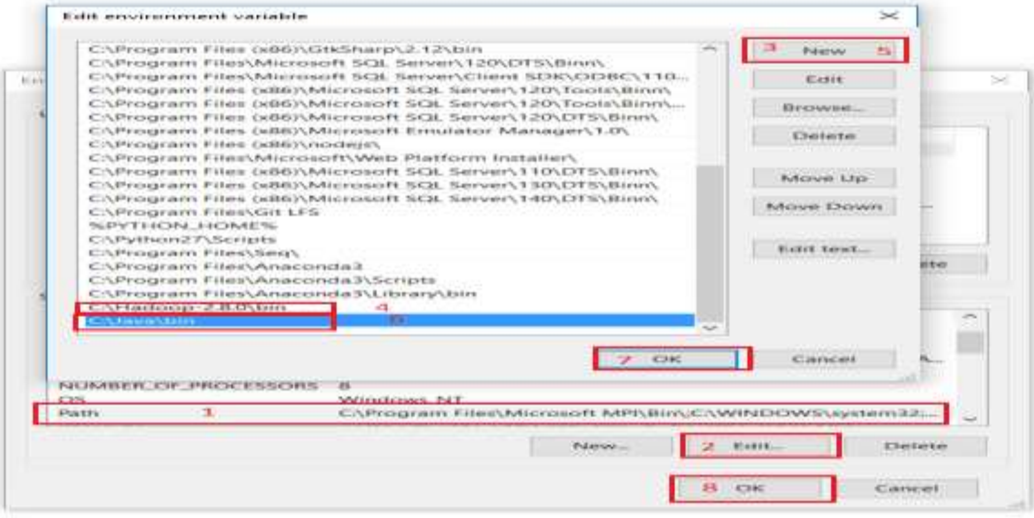


Variable name: JAVA_HOME 2
Variable value: C:\Java\bin

Variable	Value
ComSpec	C:\WINDOWS\system32\cmd.exe
DXSDK_DIR	C:\DevTools\DXSDK11\
FSHARPINSTALLDIR	C:\Program Files (x86)\Microsoft SDKs\F#\4.1\Framework\v4.0\
GIT_LFS_PATH	C:\Program Files\Git LFS
GTK_BASEPATH	C:\Program Files (x86)\GtkSharp\2.12\
HGRCPATH	C:\DevTools\Mercurial-BSI\Mercurial\mercurial.ini;%APPDATA%\Mercurial-BSI\m...
MSMPI_BIN	C:\Program Files\Microsoft MPI\Bin\

below).

5. Next we set the Hadoop bin directory path and JAVA bin directory path.



```
C:\Program Files (x86)\GtkSharp\2.12\bin
C:\Program Files\Microsoft SQL Server\120\DTS\Binn\
C:\Program Files\Microsoft SQL Server\Client SDK\ODBC\110...
C:\Program Files (x86)\Microsoft SQL Server\120\Tools\Binn\
C:\Program Files (x86)\Microsoft SQL Server\120\Tools\Binn\...
C:\Program Files (x86)\Microsoft SQL Server\120\DTS\Binn\
C:\Program Files (x86)\Microsoft Emulator Manager\1.0\
C:\Program Files (x86)\nodejs\
C:\Program Files\Microsoft\Web Platform Installer\
C:\Program Files (x86)\Microsoft SQL Server\110\DTS\Binn\
C:\Program Files (x86)\Microsoft SQL Server\130\DTS\Binn\
C:\Program Files (x86)\Microsoft SQL Server\140\DTS\Binn\
C:\Program Files\Git LFS
%PYTHON_HOME%
C:\Python27\Scripts
C:\Program Files\Seq\
C:\Program Files\Anaconda3
C:\Program Files\Anaconda3\Scripts
C:\Program Files\Anaconda3\Library\bin
C:\Hadoop-2.8.0\bin          4
C:\Java\bin                  6
```

```
NUMBER_OF_PROCESSORS   8
OS                     Windows_NT
Path          1        C:\Program Files\Microsoft MPI\Bin\;C:\WINDOWS\system32;...
```

6.

5. Configuration

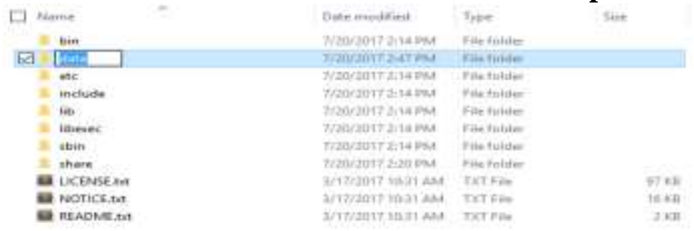
1. Edit file **C:/Hadoop-2.8.0/etc/hadoop/core-site.xml**, paste below xml paragraph and save this file.

```
<configuration>
  <property>
     <name>fs.defaultFS</name>
     <value>hdfs://localhost:9000</value>
  </property>
</configuration>
```

1. Rename "mapred-site.xml.template" to "mapred-site.xml" and edit this file **C:/Hadoop-2.8.0/etc/hadoop/mapred-site.xml**, paste below xml paragraph and save this file.

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

1. Create folder **“data”** under **“C:\Hadoop-2.8.0”**
   - Create folder **“datanode”** under **“C:\Hadoop-2.8.0\data”**
   - Create folder **“namenode”** under **“C:\Hadoop-2.8.0\data”**



Name	Date modified	Type	Size
bin	7/20/2017 2:14 PM	File folder	
data	7/20/2017 2:47 PM	File folder	
etc	7/20/2017 2:14 PM	File folder	
include	7/20/2017 2:14 PM	File folder	
lib	7/20/2017 2:14 PM	File folder	
libexec	7/20/2017 2:14 PM	File folder	
sbin	7/20/2017 2:14 PM	File folder	
share	7/20/2017 2:20 PM	File folder	
LICENSE.txt	3/17/2017 10:31 AM	TXT File	97 KB
NOTICE.txt	3/17/2017 10:31 AM	TXT File	16 KB
README.txt	3/17/2017 10:31 AM	TXT File	2 KB

   -

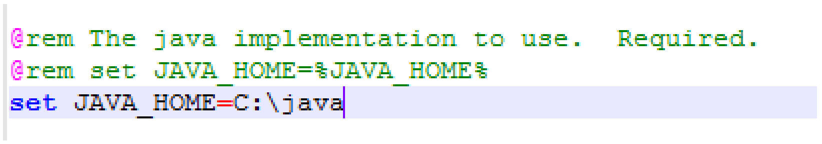
2. Edit file **C:\Hadoop-2.8.0/etc/hadoop/hdfs-site.xml**, paste below xml paragraph and save this file.

```
<configuration>
  <property>
     <name>dfs.replication</name>
     <value>1</value>
  </property>
  <property>
     <name>dfs.namenode.name.dir</name>
     <value>/hadoop-2.8.0/data/namenode</value>
  </property>
  <property>
     <name>dfs.datanode.data.dir</name>
     <value>/hadoop-2.8.0/data/datanode</value>
  </property>
</configuration>
```

1. Edit file **C:/Hadoop-2.8.0/etc/hadoop/yarn-site.xml**, paste below xml paragraph and save this file.

```
<configuration>
  <property>
        <name>yarn.nodemanager.aux-services</name>
        <value>mapreduce_shuffle</value>
  </property>
  <property>
        <name>yarn.nodemanager.auxservices.mapreduce.shuffle.class</name>
        <value>org.apache.hadoop.mapred.ShuffleHandler</value>
  </property>
</configuration>
```

1. Edit file **C:/Hadoop-2.8.0/etc/hadoop/hadoop-env.cmd** by closing the command line **"JAVA_HOME=%JAVA_HOME%"** instead of set **"JAVA_HOME=C:\Java"** (On C:\java this is path to file jdk.18.0)



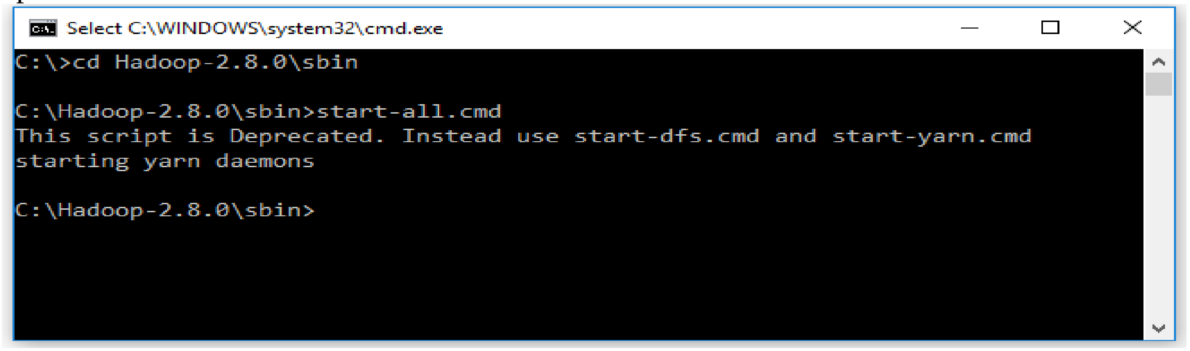
```
@rem The java implementation to use.  Required.
@rem set JAVA_HOME=%JAVA_HOME%
set JAVA_HOME=C:\java
```

6. Hadoop Configuration

1. Dowload file Hadoop Configuration.zip
2. Delete file bin on C:\Hadoop-2.8.0\bin, replaced by file bin on file just download (from Hadoop Configuration.zip).
3. Open cmd and typing command **"hdfs namenode –format"** . e

7. Testing

1. Open cmd and change directory to "C:\Hadoop-2.8.0\sbin" and type **"start-all.cmd"** to start apache.



```
C:\>cd Hadoop-2.8.0\sbin

C:\Hadoop-2.8.0\sbin>start-all.cmd
This script is Deprecated. Instead use start-dfs.cmd and start-yarn.cmd
starting yarn daemons

C:\Hadoop-2.8.0\sbin>
```

2. Make sure these apps are running
   - Hadoop Namenode
   - Hadoop datanode
   - YARN Resource ManagerYARN

Conclusion:

Hence we have performed Hadoop Installation successfully. Also studied the concept of Data node, Name node, Map Reduce and Yarn. We have studied the creation and configuration of Data node, Name node and Hadoop environment.

Assignment No. 2

Title: Hive installation, configuration and implementation

Theory:

Hive is developed on top of Hadoop. It is a data warehouse framework for querying and analysis of data that isstored in HDFS. Hive is an open source-software that lets programmers analyze large data sets on Hadoop.

The size of data sets being collected and analyzed in the industry for business intelligence is growing and in away, it is making traditional data warehousing solutions more expensive. Hadoop with MapReduce framework, is being used as an alternative solution for analyzing data sets with huge size. Though, Hadoop has proved useful for working on huge data sets, its MapReduce framework is very low level and it requires programmers to write custom programs which are hard to maintain and reuse. Hive comes here for rescue of programmers.

Hive evolved as a data warehousing solution built on top of Hadoop Map-Reduce framework.

Hive provides SQL-like declarative language, called **HiveQL**, which is used for expressing queries. UsingHive-QL users associated with SQL are able to perform data analysis very easily.

Hive engine compiles these queries into Map-Reduce jobs to be executed on Hadoop. In addition, custom Map-Reduce scripts can also be plugged into queries. Hive operates on data stored in tables which consists ofprimitive data types and collection data types like arrays and maps.

Hive comes with a command-line shell interface which can be used to create tables and execute queries.

Hive query language is similar to SQL wherein it supports subqueries. With Hive query language, it is possible to take a MapReduce joins across Hive tables. It has a support for simple **SQL like functions**- CONCAT, SUBSTR, ROUND etc., and **aggregation functions**- SUM, COUNT, MAX etc. It also supports GROUP BY and SORT BY clauses. It is also possible to write user defined functions in Hive query language.

- Hive Vs Map Reduce

Prior to choosing one of these two options, we must look at some of their features.

While choosing between Hive and Map reduce following factors are taken in consideration;

- Type of Data
- Amount of Data
- Complexity of Code

Difference Between Hive and Map Reduce:

Feature	**Hive**	**Map Reduce**
Language	It Supports SQL like query language for interaction and forData modeling	It compiles language with two main taskspresent in it. One is map task, and another one is a reducer. We can define these task using Java orPython
Level of abstraction	Higher level of Abstraction on top of HDFS	Lower level of abstraction
Efficiency in Code	Comparatively lesser than Mapreduce	Provides High efficiency
Extent of code	Less number of lines code requiredfor execution	More number of lines of codes to be defined
Type of Development work required	Less Development work required	More development work needed

Exercise:

```
#!/usr/bin/env python

import sys

from hive import ThriftHive
from hive.ttypes import HiveServerException
from thrift import Thrift
from thrift.transport import TSocket
from thrift.transport import TTransport
from thrift.protocol import TBinaryProtocol

try:
    transport = TSocket.TSocket('localhost', 10000)
    transport = TTransport.TBufferedTransport(transport)
    protocol = TBinaryProtocol.TBinaryProtocol(transport)
     client = ThriftHive.Client(protocol)
    transport.open()

    client.execute("CREATE TABLE r(a STRING, b INT, c DOUBLE)")
    client.execute("LOAD TABLE LOCAL INPATH '/path' INTO TABLE r")
    client.execute("SELECT * FROM r")
    while (1):
      row = client.fetchOne()
      if (row == None):
        break
      print row
    client.execute("SELECT * FROM r")
    print client.fetchAll()
    transport.close()

except Thrift.TException, tx:
    print '%s' % (tx.message)
```

CONCLUSION :

Hive installation and configuration on Hadoop is completed successfully.

Assignment No 3

Title: SQL and NoSQL programming for Data Science

Theory:

Structured Query Language (SQL)
SQL is a powerful programming language used to manipulate data in a relational database management system (RDBMS). SQL is relatively easy, yet so powerful and efficient. Developers and data scientists use SQL to add, delete, update, or perform specific o[eration on a relational database.
SQL is not just for performing simple operations on databases; it can also be used to design databases or perform some analytics of the data stored.
Why SQL?
SQL is very popular, and it's widely used in software development — in general — and data science in particular for various reasons, including:

1. **Flexibility:** SQL allows you to add or delete new columns, tables, rename relations, and make other changes while the database is up and running, and queries are happening. Moreover, it can be integrated into many script languages hassle-free.
2. **Ease of use:** Learning the basics of SQL is simple and straight-forward. There is no confusing syntax of hidden tips to master using SQL.
3. **No redundancy:** Because of the relational nature of SQL, you can have all the information you need about an entry all in the same location, so you will not need to repeat the same information across all tables.
4. **Reliability:** Most relational databases are accessible to export and import, making backup and restore a breeze. These exports can be performed while the database is running, making restore on failure easy.

ACID Properties:
A—Atomicity
The property which guarantees atomic operations, either a set of queries can complete as a whole or none does.This is the key feature for transactions.

C—Consistency
Data are available as soon as they are completely inserted or updated.

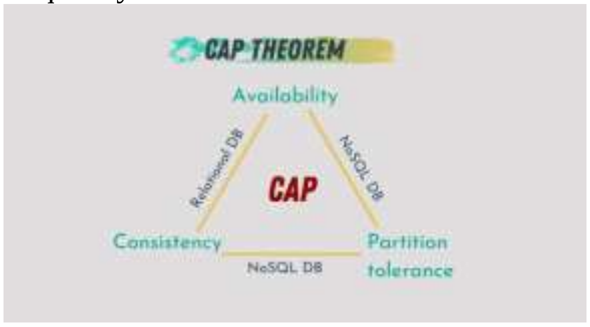
I—Isolation
Implies that, transactions are independent. Therefore data will not be negatively affected by two transactionshappening on same set of data.

D—Durability
Committed data after a transaction or any other operation is never lost. Either they get inserted or failure isnotified (Failed transactions).

SQL vs. no-SQL databases

Whenever you are assigned a new project or attempt to design a w database, the first question you probably ask yourself is *"which database should I use? SQL or NoSQL?"*.
Here's the thing, when trying to choose a correct database type, I often refer to the CAP theorem. The CAP theorem describes the relationship between three aspects of your database: availability, consistency, and partition tolerance.

1. **Consistency:** This means that every inquiry to the database should return the most recent value. There are 5 different levels of consistency, from strong — immediate — to eventual — out-of-date results.
2. **Availability:** This means anyone can make a request for data and get a response, even if one or more items of the database are down.
3. **Partition tolerance:** A *partition* is a communications break within a system. Tolerance means the database should function adequately even if the communications between aspects of it are broken.



NoSQL database is more and more popular in the modern data architecture. It has become a powerful way to store data in a specialized format that yields fast performance for a large amount of data. There have been many NoSQL databases available on the market, while new ones are still emerging. The most popular categorization consists of 4 types: Wide Column, Document, Key-value Pairs, and Graph. Among many NoSQL databases, below lists a few popular ones:

- Wide Columnar: Cassandra, HBase, AWS DynamoDB
- Document: Couchbase, MongoDB, Azure Cosmos DB, AWS DynamoDB
- Graph: Neo4J, Azure Cosmos DB, TigerGraph, AWS Neptune
- Key-Value Pairs: AWS Dynamo, Redis, Oracle NoSQL

NoSQL databases, therefore, are designed to offer the flexibility of adding information and fast performance when data volume is large, while sacrificing the three Normal Forms and tolerating the risks of data redundancy and insert/update/delete abnormalities. For the same reason, a NoSQL database is best for semi-structured or unstructured data that do not fit relational database designs and data scenarios that require few constraints to enforce its accuracy. Furthermore, most NoSQL databases are designed for fast read performance with the trade of slowing down frequent updates, inserts, and deletions.

MongoDB is a document-oriented NoSQL **database**. It differs from relational databases in its flexibility and performance. Find out everything you need to know about this must-have tool for data engineering.
MongoDB is a document-oriented NoSQL database that appeared in the mid-2000s. It is used for storing massive volumes of data.

Unlike a traditional **SQL relational** database, MongoDB does not rely on tables and columns. Data is stored as collections and documents.

Documents are value/key pairs that serve as the basic data unit. Collections contain sets of documents and functions. They are the equivalent of tables in classical relational databases.

The characteristics of MongoDB

Each MongoDB database contains collections, themselves containing documents. Each document is different and can have a variable number of fields. The size and content of each document also vary.

The structure of a document corresponds to the way developers build their classes and objects in the programming language used. In general, classes are not rows and columns but have a clear structure consisting of value/key pairs.

Documents do not have a predefined schema and fields can be added at will. The data model available within MongoDB makes it easier to represent hierarchical relationships or other complex structures.

Another major feature of MongoDB is the **elasticity of its environments**. Many companies have clusters of over 100 nodes for databases containing millions of documents.

A collection is a **group of MongoDB documents**, and corresponds to a table created with any other RDMS like Oracle or MS SQL on a relational database. It has no predefined structure.

A database is a **container of collections**, just as an RDMS is a container of tables for relational databases. Each has its own set of files on the file system. A MongoDB server can store multiple databases.

Finally, **JSON (JavaScript Object Notation)** is a plain text format for expressing structured data. It is supported by many programming languages.

Why use MongoDB ? What are the advantages ?

MongoDB has several major advantages. First of all, this document-oriented NoSQL database is **very flexible** and adapted to the concrete use cases of an enterprise.

Ad hoc queries allow you to find specific fields within documents. It is also possible to create indexes to improve search performance. Any field can be indexed.

Another advantage is the ability **to create "replica sets"** consisting of two or more MongoDB instances. Each member can act as a secondary or primary replica at any time.

The primary replica is the main server, which interacts with the client and performs all read and write operations. The secondary replicas keep a copy of the data. Thus, in case of failure of the primary replica, the switchover to the secondary is done automatically. This system guarantees high availability.

Finally, **the concept of sharding** allows for horizontal scaling by distributing the data among multiple MongoDB instances. The database can be run on multiple servers, and this allows load balancing or duplicating data to keep the system functional in case of hardware failure.

Because of these many advantages, MongoDB is now a widely used tool in the field of data engineering.

Exercise:

```
import mysql.connector
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password=""
)

print(mydb)
<mysql.connector.connection.MySQLConnection object at 0x0000019E7A385D68>
mycursor = mydb.cursor()
x=mycursor.execute("CREATE DATABASE candidates")


mycursor = mydb.cursor()

mycursor.execute("SHOW DATABASES")

for x in mycursor:
  print(x)
('a',)
('b',)
('c',)
('candidates',)
('information_schema',)
('mydatabase',)
('mysql',)
('performance_schema',)
('phpmyadmin',)
('student',)
('studentdata',)
('test',)
mydb = mysql.connector.connect(
  host="localhost",
  user="root",
  password="",
  database="candidates"
)
#creating a cursor to hit the mysql
mycursor = mydb.cursor()
mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE studentinfo (id INT AUTO_INCREMENT PRIMARY KE
Y, name VARCHAR(255), address VARCHAR(255))")
```

```
#check the created table
mycursor.execute("SHOW TABLES")
for x in mycursor:
    print(x)

('studentinfo',)
mycursor = mydb.cursor()

mycursor.execute("CREATE TABLE studentdata (roll_no INT AUTO_INCREMENT PRIMAR
Y KEY, phone INT, email VARCHAR(255))")
#check the created table
mycursor.execute("SHOW TABLES")
for x in mycursor:
    print(x)

('studentdata',)
('studentinfo',)
#alter the table schema, add a coloumn to table
mycursor.execute("ALTER TABLE studentdata ADD COLUMN registration_id INT")

#inserting values to tables

mycursor = mydb.cursor()

sql = "INSERT INTO studentinfo (name, address) VALUES (%s, %s)"
val = ("John", "Highway 21")
mycursor.execute(sql, val)

print(mycursor.rowcount, "record inserted.")
1 record inserted.
sql = "INSERT INTO studentinfo (name, address) VALUES (%s, %s)"
val = ("Joe", "backyard ")
mycursor.execute(sql, val)

print(mycursor.rowcount, "record inserted.")
1 record inserted.
sql = "INSERT INTO studentinfo (name, address) VALUES (%s, %s)"
val = ("Jack", "Salisbury Park")
mycursor.execute(sql, val)

print(mycursor.rowcount, "record inserted.")
1 record inserted.
sql = "INSERT INTO studentinfo (name, address) VALUES (%s, %s)"
val = ('Amy', 'Apple st 652')
```

```
mycursor.execute(sql, val)

print(mycursor.rowcount, "record inserted.")
1 record inserted.
mycursor = mydb.cursor()
sql = "INSERT INTO studentinfo (name, address) VALUES (%s, %s)"
val =[
  ('Peter', 'Lowstreet 4'),
  ('Amy', 'Apple st 652'),
  ('Hannah', 'Mountain 21'),
  ('Michael', 'Valley 345'),
  ('Sandy', 'Ocean blvd 2'),
  ('Betty', 'Green Grass 1'),
  ('Richard', 'Sky st 331'),
  ('Susan', 'One way 98'),
  ('Vicky', 'Yellow Garden 2'),
  ('Ben', 'Park Lane 38'),
  ('William', 'Central st 954'),
  ('Chuck', 'Main Road 989'),
  ('Viola', 'Sideway 1633')
]
mycursor.executemany(sql, val)

print(mycursor.rowcount, "was inserted.")
13 was inserted.
#commit changes to the database
mydb.commit()

#view the data in tables
mycursor.execute("SELECT * FROM studentinfo")
for x in mycursor:
    print(x)

(1, 'John', 'Highway 21')
(2, 'Joe', 'backyard ')
(3, 'Jack', 'Salisbury Park')
(4, 'Peter', 'Lowstreet 4')
(5, 'Amy', 'Apple st 652')
#update statment
sql = "UPDATE studentinfo SET name = 'Shruti' WHERE id = 1"
mycursor.execute(sql)
print(mycursor.rowcount, "record(s) affected")

1 record(s) affected
```

```
#view the data in tables
mycursor.execute("SELECT * FROM studentinfo")
for x in mycursor:
   print(x)
(1, 'Shruti', 'Highway 21')
(2, 'Joe', 'backyard ')
(3, 'Jack', 'Salisbury Park')
(4, 'Peter', 'Lowstreet 4')
(5, 'Amy', 'Apple st 652')
#DELETE the data
sql = "DELETE FROM studentinfo WHERE name = 'Joe'"
mycursor.execute(sql)
mydb.commit()
print(mycursor.rowcount, "record(s) deleted")
1 record(s) deleted

#view the data in tables
mycursor.execute("SELECT * FROM studentinfo")
for x in mycursor:
   print(x)
#order by name
sql = "SELECT * FROM studentinfo ORDER BY name DESC"
mycursor.execute(sql)
for x in mycursor:
   print(x)

#order by name
sql = "SELECT * FROM studentinfo ORDER BY name "
mycursor.execute(sql)
for x in mycursor:
   print(x)
#Drop table
sql = "DROP TABLE studentdata"
mycursor.execute(sql)
#check the dropped table
mycursor.execute("SHOW TABLES")
```

3B

```
import pymongo
myclient = pymongo.MongoClient("mongodb://localhost:27017/")

mydb = myclient["mydatabase"]
print(myclient.list_database_names())
['admin', 'config', 'local', 'studentData']
```

```
#create a collection
mydb = myclient["mydatabase"]
mycol = mydb["customers"]
#print collections
print(mydb.list_collection_names())
[]
mydict = { "name": "John", "address": "Highway 37" }
x = mycol.insert_one(mydict)
print(x)
<pymongo.results.InsertOneResult object at 0x0000028A707C2688>
mydict = { "name": "Peter", "address": "Lowstreet 27" }
x = mycol.insert_one(mydict)
print(x.inserted_id)

606ecbc9c3a2ecb3c9f250a8
#insert a list
mylist = [
{ "name": "Amy", "address": "Apple st 652"},
{ "name": "Hannah", "address": "Mountain21"},
{ "name": "Michael", "address": "Valley345"},
{ "name": "Sandy", "address": "Ocean blvd2"},
{ "name": "Betty", "address": "Green Grass 1"},
{ "name": "Richard", "address": "Sky st 331"},
{ "name": "Susan", "address": "One way 98"},
{ "name": "Vicky", "address": "Yellow Garden 2"},
{ "name": "Ben", "address": "Park Lane 38"},
{ "name": "William", "address": "Central st 954"},
{ "name": "Chuck", "address": "Main Road 989"},
{ "name": "Viola", "address": "Sideway 1633"}
]
x = mycol.insert_many(mylist)
#print list of the _id values of the inserted documents:
print(x.inserted_ids)

[ObjectId('606ecbddc3a2ecb3c9f250a9'), ObjectId('606ecbddc3a2ecb3c9f250aa'), ObjectId('606
ecbddc3a2ecb3c9f250ab'), ObjectId('606ecbddc3a2ecb3c9f250ac'), ObjectId('606ecbddc3a2ecb3
c9f250ad'), ObjectId('606ecbddc3a2ecb3c9f250ae'), ObjectId('606ecbddc3a2ecb3c9f250af'), Obj
ectId('606ecbddc3a2ecb3c9f250b0'), ObjectId('606ecbddc3a2ecb3c9f250b1'), ObjectId('606ecbd
dc3a2ecb3c9f250b2'), ObjectId('606ecbddc3a2ecb3c9f250b3'), ObjectId('606ecbddc3a2ecb3c9f2
50b4')]

#select one data from a collection in MongoDB
x = mycol.find_one()
x
```

```
{'_id': ObjectId('606ecb9cc3a2ecb3c9f250a6'),
 'name': 'John',
 'address': 'Highway 37'}
#select all data from a collection in MongoD
for x in mycol.find():
    print(x)
{'_id': ObjectId('606ecb9cc3a2ecb3c9f250a6'), 'name': 'John', 'address': 'Highway 37'}
{'_id': ObjectId('606ecbaec3a2ecb3c9f250a7'), 'name': 'John', 'address': 'Highway 37'}
{'_id': ObjectId('606ecbc9c3a2ecb3c9f250a8'), 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250a9'), 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250aa'), 'name': 'Hannah', 'address': 'Mountain21'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250ab'), 'name': 'Michael', 'address': 'Valley345'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250ac'), 'name': 'Sandy', 'address': 'Ocean blvd2'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250ad'), 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250ae'), 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250af'), 'name': 'Susan', 'address': 'One way 98'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b0'), 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b1'), 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b2'), 'name': 'William', 'address': 'Central st 954'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b3'), 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b4'), 'name': 'Viola', 'address': 'Sideway 1633'}
#find by query
myquery = { "address": "Park Lane 38" }
mydoc = mycol.find(myquery)
for x in mydoc:
    print(x)
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b1'), 'name': 'Ben', 'address': 'Park Lane 38'}
#sort by names
mydoc = mycol.find().sort("name")
for x in mydoc:
     print(x)
{'_id': ObjectId('606ecbddc3a2ecb3c9f250a9'), 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b1'), 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250ad'), 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b3'), 'name': 'Chuck', 'address': 'Main Road 989'}
#delete one document
myquery = { "address": "Mountain 21" }
mycol.delete_one(myquery)
<pymongo.results.DeleteResult at 0x28a707f26c8>
#update sigle value
myquery = { "address": "Valley 345" }
newvalues = { "$set": { "address": "Canyon 123" } }
mycol.update_one(myquery,newvalues)
#print "customers" after the update:
```

```
for x in mycol.find():
    print(x)
{'_id': ObjectId('606ecb9cc3a2ecb3c9f250a6'), 'name': 'John', 'address': 'Highway 37'}
{'_id': ObjectId('606ecbaec3a2ecb3c9f250a7'), 'name': 'John', 'address': 'Highway 37'}
{'_id': ObjectId('606ecbc9c3a2ecb3c9f250a8'), 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250a9'), 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250aa'), 'name': 'Hannah', 'address': 'Mountain21'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250ab'), 'name': 'Michael', 'address': 'Valley345'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250ac'), 'name': 'Sandy', 'address': 'Ocean blvd2'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250ad'), 'name': 'Betty', 'address': 'Green Grass 1'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250ae'), 'name': 'Richard', 'address': 'Sky st 331'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250af'), 'name': 'Susan', 'address': 'One way 98'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b0'), 'name': 'Vicky', 'address': 'Yellow Garden 2'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b1'), 'name': 'Ben', 'address': 'Park Lane 38'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b2'), 'name': 'William', 'address': 'Central st 954'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b3'), 'name': 'Chuck', 'address': 'Main Road 989'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250b4'), 'name': 'Viola', 'address': 'Sideway 1633'}
#update multiple values
myquery = { "address": { "$regex": "^S" } }
newvalues = { "$set": { "name": "Minnie" } }
x = mycol.update_many(myquery, newvalues)
x
<pymongo.results.UpdateResult at 0x28a707f2288>
#select five coloumns
myresult = mycol.find().limit(5)
#print the result:
for x in myresult:
    print(x)
{'_id': ObjectId('606ecb9cc3a2ecb3c9f250a6'), 'name': 'John', 'address': 'Highway 37'}
{'_id': ObjectId('606ecbaec3a2ecb3c9f250a7'), 'name': 'John', 'address': 'Highway 37'}
{'_id': ObjectId('606ecbc9c3a2ecb3c9f250a8'), 'name': 'Peter', 'address': 'Lowstreet 27'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250a9'), 'name': 'Amy', 'address': 'Apple st 652'}
{'_id': ObjectId('606ecbddc3a2ecb3c9f250aa'), 'name': 'Hannah', 'address': 'Mountain21'}
#drop the collection
x=mycol.drop()
print(x)
None
```

Conclusion:

We have studied SQL and NOSQL programming language along with ACID properties, CAP theorem and CURD operation. We have performed the different SQL commands Insert, update, drop. Also performed NoSQL commands.

Assignment 4

Title: Understanding data formats of Numpy: ndarrays (1D, 2D & 3D arrays), Array creation routines

Theory:

What is NumPy?
NumPy is a general-purpose array-processing package. It provides a high-performance multidimensional array object, and tools for working with these arrays.

It is the fundamental package for scientific computing with Python. It contains various features including these important ones:

A powerful N-dimensional array object
Sophisticated (broadcasting) functions
Tools for integrating C/C++ and Fortran code
Useful linear algebra, Fourier transform, and random number capabilities
Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.
Arbitrary data-types can be defined using Numpy which allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

Exercise:

```
# Python program to demonstrate
# basic array characteristics
import numpy as np

# Creating array object
arr = np.array( [[ 1, 2, 3],
                [ 4, 2, 5]] )

# Printing type of arr object
print("Array is of type: ", type(arr))

# Printing array dimensions (axes)
print("No. of dimensions: ", arr.ndim)

# Printing shape of array
print("Shape of array: ", arr.shape)

# Printing size (total number of elements) of array
print("Size of array: ", arr.size)
```

```
# Printing type of elements in array
print("Array stores elements of type: ", arr.dtype)

Output :
Array is of type:
No. of dimensions:  2
Shape of array:  (2, 3)
Size of array:  6
Array stores elements of type:  int64
# array creation techniques
import numpy as np

# Creating array from list with type float
a = np.array([[1, 2, 4], [5, 8, 7]], dtype = 'float')
print ("Array created using passed list:\n", a)

# Creating array from tuple
b = np.array((1 , 3, 2))
print ("\nArray created using passed tuple:\n", b)

# Creating a 3X4 array with all zeros
c = np.zeros((3, 4))
print ("\nAn array initialized with all zeros:\n", c)

# Create a constant value array of complex type
d = np.full((3, 3), 6, dtype = 'complex')
print ("\nAn array initialized with all 6s."
        "Array type is complex:\n", d)

# Create an array with random values
e = np.random.random((2, 2))
print ("\nA random array:\n", e)

# Create a sequence of integers
# from 0 to 30 with steps of 5
f = np.arange(0, 30, 5)
print ("\nA sequential array with steps of 5:\n", f)

# Create a sequence of 10 values in range 0 to 5
g = np.linspace(0, 5, 10)
print ("\nA sequential array with 10 values between"
                                "0 and 5:\n", g)
```

```
# Reshaping 3X4 array to 2X2X3 array
arr = np.array([[1, 2, 3, 4],
                [5, 2, 4, 2],
                [1, 2, 0, 1]])

newarr = arr.reshape(2, 2, 3)
print ("\nOriginal array:\n", arr)
print ("Reshaped array:\n", newarr)
```

Slicing: Just like lists in python, NumPy arrays can be sliced. As arrays can be multidimensional, you need to specify a slice for each dimension of the array.
Integer array indexing: In this method, lists are passed for indexing for each dimension. One to one mapping of corresponding elements is done to construct a new arbitrary array.
Boolean array indexing: This method is used when we want to pick elements from array which satisfy some condition.

```
# Python program to demonstrate
# indexing in numpy
import numpy as np

# An exemplar array
arr = np.array([[-1, 2, 0, 4],
                [4, -0.5, 6, 0],
                [2.6, 0, 7, 8],
                [3, -7, 4, 2.0]])

# Slicing array
temp = arr[:2, ::2]
print ("Array with first 2 rows and alternate"
                    "columns(0 and 2):\n", temp)

# Integer array indexing example
temp = arr[[0, 1, 2, 3], [3, 2, 1, 0]]
print ("\nElements at indices (0, 3), (1, 2), (2, 1),"
                                "(3, 0):\n", temp)

# boolean array indexing example
cond = arr > 0 # cond is a boolean array
temp = arr[cond]
print ("\nElements greater than 0:\n", temp)
```

Output :

```
Array with first 2 rows and alternatecolumns(0 and 2):
 [[-1.  0.]
 [ 4.  6.]]

Elements at indices (0, 3), (1, 2), (2, 1),(3, 0):
 [ 4.  6.  0.  3.]

Elements greater than 0:
 [ 2.   4.   4.   6.   2.6  7.   8.   3.   4.   2. ]
```

4. Basic operations: Plethora of built-in arithmetic functions are provided in NumPy.

Operations on single array: We can use overloaded arithmetic operators to do element-wise operation on array to create a new array. In case of +=, -=, *= operators, the existing array is modified.

```
# Python program to demonstrate
# basic operations on single array
import numpy as np

a = np.array([1, 2, 5, 3])

# add 1 to every element
print ("Adding 1 to every element:", a+1)

# subtract 3 from each element
print ("Subtracting 3 from each element:", a-3)

# multiply each element by 10
print ("Multiplying each element by 10:", a*10)

# square each element
print ("Squaring each element:", a**2)

# modify existing array
a *= 2
print ("Doubled each element of original array:", a)

# transpose of array
a = np.array([[1, 2, 3], [3, 4, 5], [9, 6, 0]])

print ("\nOriginal array:\n", a)
print ("Transpose of array:\n", a.T)
```

```
Output :
Adding 1 to every element: [2 3 6 4]
Subtracting 3 from each element: [-2 -1  2  0]
Multiplying each element by 10: [10 20 50 30]
Squaring each element: [ 1  4 25  9]
Doubled each element of original array: [ 2  4 10  6]

Original array:
 [[1 2 3]
[3 4 5]
 [9 6 0]]
Transpose of array:
 [[1 3 9]
 [2 4 6]
 [3 5 0]]
```

Unary operators: Many unary operations are provided as a method of ndarray class. This includes sum, min, max, etc. These functions can also be applied row-wise or column-wise by setting an axis parameter.

```
# Python program to demonstrate
# unary operators in numpy
import numpy as np

arr = np.array([[1, 5, 6],
                [4, 7, 2],
                [3, 1, 9]])

# maximum element of array
print ("Largest element is:", arr.max())
print ("Row-wise maximum elements:",
                    arr.max(axis = 1))

# minimum element of array
print ("Column-wise minimum elements:",
                        arr.min(axis = 0))

# sum of array elements
print ("Sum of all array elements:",
                           arr.sum())

# cumulative sum along each row
print ("Cumulative sum along each row:\n",
                        arr.cumsum(axis = 1))
```
Output :

```
Largest element is: 9
Row-wise maximum elements: [6 7 9]
Column-wise minimum elements: [1 1 2]
Sum of all array elements: 38
Cumulative sum along each row:
[[ 1  6 12]
 [ 4 11 13]
 [ 3  4 13]]
```
Binary operators: These operations apply on array elementwise and a new array is created. You can use all basic arithmetic operators like +, -, /, , etc. In case of +=, -=, = operators, the existing array is modified.

```
# Python program to demonstrate
# binary operators in Numpy
import numpy as np

a = np.array([[1, 2],
            [3, 4]])
b = np.array([[4, 3],
            [2, 1]])

# add arrays
print ("Array sum:\n", a + b)

# multiply arrays (elementwise multiplication)
print ("Array multiplication:\n", a*b)

# matrix multiplication
print ("Matrix multiplication:\n", a.dot(b))
```
Output:

```
Array sum:
[[5 5]
 [5 5]]
Array multiplication:
[[4 6]
 [6 4]]
Matrix multiplication:
[[ 8  5]
 [20 13]]
```

```
#numpy.ones : returns a new array of specified size and type ,filled with ones
y=np.ones(4)#by default it is float
print(y)
y=np.ones([2,4],dtype=int)
print(y)
y=np.ones([2,4],dtype=float)
print(y)
y=np.ones([2,4],dtype=bool)
print(y)
```

output:

```
[1. 1. 1. 1.]
[[1 1 1 1]
 [1 1 1 1]]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]]
[[ True  True  True  True]
 [ True  True  True  True]]
```

```
#numpy.zeros : returns a new array of specified size and type ,filled with zeros
y=np.zeros(4)#by default it is float
print(y)
y=np.zeros([2,4],dtype=int)
print(y)
y=np.zeros([2,4],dtype=float)
print(y)
y=np.zeros([2,4],dtype=bool)
print(y)
```

output:

```
[0. 0. 0. 0.]
[[0 0 0 0]
 [0 0 0 0]]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[[False False False False]
 [False False False False]]
```

```
#np.asarray it is used for creating a python sequence to an array
x=[1,2,3,4]
a13=np.asarray(x)
print (a13)
```

output:

```
[1 2 3 4]
[1. 2. 3. 4.]
```

```
#numpy.fromiter : this function builds an ndarray object from any iterable object
#a new one dimensional array is returned by this function
list=range(5)
it=iter(list)

x=np.fromiter(it,dtype=float)
print(x)
```
output:
```
[0. 1. 2. 3. 4.]
```

```
#numpy.arrange : This Function returns an ndarray object containing evenly spaced values
within the given range
x=np.arange(5)
print(x)
x=np.arange(4,dtype=float)#dtype parameter
print(x)
x=np.arange(10,20,2)#star and stop parameter with steps of jump
print(x)
x=np.arange(10,20,3)
print(x)
```
output:
```
[0 1 2 3 4]
[0. 1. 2. 3.]
[10 12 14 16 18]
[10 13 16 19]
```
```
#numpy.linspace : This function is similar to arange function.In this function,instead of size ,
#the no. of evenly spaced values between the interval is specified
x=np.linspace(1,2,5,retstep=True)
#If retstep is true ,returns sample and step between the consecutive numbers
print(x)
x=np.linspace(1,5,5,retstep=True)
#If retstep is true ,returns sample and step between the consecutive numbers
print(x)
x=np.linspace(2,12,6,retstep=True)

#If retstep is true ,returns sample and step between the consecutive numbers
print(x)
```
output:
```
(array([1.  , 1.25, 1.5 , 1.75, 2.  ]), 0.25)
(array([1., 2., 3., 4., 5.]), 1.0)
(array([ 2.,  4.,  6.,  8., 10., 12.]), 2.0)
```
In [158]:

```
#numpy.logspace : This Function returns an ndarray object that contains the number that are
evenly spaced on a log scale
# Start and Stop Endpoints of the scale are indices of the base ,usually 10
#default base is 10
b4=np.logspace(1.0,2.0,num=10)
print(b4)
b5=np.logspace(1,3,num=10,base=2)
print(b5)
```

output:

```
[ 10.          12.91549665  16.68100537  21.5443469   27.82559402
  35.93813664  46.41588834  59.94842503  77.42636827 100.        ]
[2.         2.33305808 2.72158    3.1748021  3.70349885 4.32023896
 5.0396842  5.87893797 6.85795186 8.        ]
```

```
#3D Array Indexing
b6=np.array([
    [[1,2,3],[11,4,5]],
    [[14,15,6],[7,8,9]]
])
print(b6[0][1][2])
b6=np.array([
    [[1,2,3],[4,5,6],[7,8,9]],
    [[10,11,12],[13,14,15],[16,17,18]]
])
print(b6[0][2][2])
b6=np.array([
    [[1,2,3],[4,5,6],[7,8,9],[101,102,103]],
    [[10,11,12],[13,14,15],[16,17,18],[201,202,203]],
    [[20,21,22],[23,24,25],[26,27,28],[301,302,303]],
    [[30,31,32],[33,34,35],[36,37,38],[401,402,403]],
])
print(b6[2][3][1])
```

output:

```
5
9
302
```

```
#1D Array Indexing
b7=np.array([1,2,3])
print(b7[1])
#2D Array Indexing
b8=np.array([[1,2,3],[3,4,5],[4,5,6]])
print(b8[1][2])
#extra solving
```

```
b9=np.array([[[1,2,3,4],[5,6,7,8]],
            [[11,12,13,15],[16,17,18,19]]
           ])
print(b9[1][1][2])
```

output:

```
2
5
18
```

```
#slicing of arrays
#2D array
b10=np.array([[10,11,12,13,14],[15,16,17,18,19],[20,21,22,23,24],[25,26,27,28,29]])
print(b10[1:,2:4])
print(b10[:,4:])
print(b10[:3,:3])
```

output:

```
[[17 18]
 [22 23]
 [27 28]]
[[14]
 [19]
 [24]
 [29]]
[[10 11 12]
 [15 16 17]
 [20 21 22]]
```

```
#3D Array Slicing
c1=np.array([[[10,11,12],[13,14,15],[16,17,18]],
            [[20,21,22],[23,24,25],[26,27,28]],
            [[30,31,32],[33,34,35],[36,37,38]]])
#picking a row or column in a 3D Array
print(c1[1,2])
print(c1[0,:,1])
print(c1[:,1,2])
print(c1[:,1:3,:])
```

output:

```
[26 27 28]
[11 14 17]
[15 25 35]
[[[13 14 15]
  [16 17 18]]
[[23 24 25]
  [26 27 28]]

 [[33 34 35]
  [36 37 38]]]
```

Conclusion:

We have performed various operations related to Numpy such as creation of 1D, 2D and 3D array. Along with that we performed addition and Deletion of row, columns, slicing and I ndexing operations with 1D, 2D and 3D array

Assignment 5

Title: Understanding data formats of Pandas: Series, Dataframe, Panel; Creating, Appending, Deleting. Importing different types of Datasets.

Theory:
Pandas contains high-level data structures and manipulation tools designed to make data analysis fast and easyin Python. pandas is built on top of NumPy and makes it easy to use in NumPy-centric applications.

Pandas deals with the following three data structures −

- Series
- DataFrame
- Panel

These data structures are built on top of Numpy array, which means they are fast.

Code and Output of Each Statement :

#Create an Empty Series

```
In [1]:
import pandas as pd
s= pd.Series()
print(s)
Series([], dtype: float64)
In [2]:
```

#Create a Series from ndarray

```
In [3]:
import numpy as np
data = np.array(['a','b','c','d'])
print(data)
['a' 'b' 'c' 'd']
In [4]:
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[10,20,30,40])
print(s)
10    a
20    b
30    c
40    d
dtype: object
In [5]:
```

#Create a Series from dict#

```
In [6]:
import pandas as pd
```

```
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data,index=[1,2,3])
print(s)
1   NaN
2   NaN
3   NaN
dtype: float64
In [8]:
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data,index=['b','c','d','a'])
print(s)
b    1.0
c    2.0
d    NaN
a    0.0
dtype: float64
In [9]:
#Create a Series from Scalar
In [10]:
import pandas as pd
import numpy as np
s = pd.Series(3, index=[0, 1, 2, 3,4,5,6])
print (s)
0    3
1    3
2    3
3    3
4    3
5    3
6    3
dtype: int64
In [11]:
#Accessing Data from Series with Position
In [12]:
import pandas as pd
s = pd.Series([11,25,37,89,95],index = ['a','b','c','d','e'])
print(s)
#retrieve the first element
print(s[0])
print(s[3])
a    11
```

```
b    25
c    37
d    89
e    95
dtype: int64
11
89
In [7]:
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first three element
print(s[:3])
a    1
b    2
c    3
dtype: int64
In [8]:
print(s[-3:])
c    3
d    4
e    5
dtype: int64
In [9]:
#Retrieve Data Using Label (Index)
In [10]:
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
print(s)
#retrieve a single element
print(s['c'])
a    1
b    2
c    3
d    4
e    5
dtype: int64
3
In [11]:
print(s['f'])
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_value(self, series, key)
   4380             try:
```

```
-> 4381                return libindex.get_value_box(s, key)
   4382            except IndexError:

pandas/_libs/index.pyx in pandas._libs.index.get_value_box()

pandas/_libs/index.pyx in pandas._libs.index.get_value_at()

pandas/_libs/util.pxd in pandas._libs.util.get_value_at()

pandas/_libs/util.pxd in pandas._libs.util.validate_indexer()

TypeError: 'str' object cannot be interpreted as an integer

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<ipython-input-11-facc49cb689c> in <module>
----> 1 print(s['f'])

~\Anaconda3\lib\site-packages\pandas\core\series.py in __getitem__(self, key)
    866         key = com.apply_if_callable(key, self)
    867         try:
--> 868             result = self.index.get_value(self, key)
    869 
    870             if not is_scalar(result):

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_value(self, series, key)
   4387                     raise InvalidIndexError(key)
   4388                 else:
-> 4389                     raise e1
   4390             except Exception:  # pragma: no cover
   4391                 raise e1

~\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_value(self, series, key)
   4373         try:
   4374             return self._engine.get_value(s, k,
-> 4375                                           tz=getattr(series.dtype, 'tz', None))
   4376         except KeyError as e1:
   4377             if len(self) > 0 and (self.holds_integer() or self.is_boolean()):

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_value()

pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_value()
```

```
pandas/_libs/index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item(
)

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item(
)

KeyError: 'f'
```

In [12]:

#Create an Empty DataFrame

In [13]:

```
import pandas as pd
df = pd.DataFrame()
print(df)
print(df.empty)
```

```
Empty DataFrame
Columns: []
Index: []
True
```

In [14]:

#Create a DataFrame from Lists

In [15]:

```
import pandas as pd
data =[[1,2,3,4,5],[1,1,1,1,1,],[2,2,2,2,2]]
l=[1,2,3,4,5]
df = pd.DataFrame(data)
df1=pd.DataFrame(l)
print(df)
print(df1)
```

```
   0  1  2  3  4
0  1  2  3  4  5
1  1  1  1  1  1
2  2  2  2  2  2
   0
0  1
1  2
2  3
3  4
4  5
```

In [16]:

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
```

```
print(df)
     Name  Age
0    Alex   10
1     Bob   12
2  Clarke   13
```

In [17]:

#Create a DataFrame from Dict of ndarrays / Lists

In [18]:

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print(df)
    Name  Age
0    Tom   28
1   Jack   34
2  Steve   29
3  Ricky   42
```

In [19]:

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])
print(df)
        Name  Age
rank1    Tom   28
rank2   Jack   34
rank3  Steve   29
rank4  Ricky   42
```

In [20]:

#Create a DataFrame from List of Dicts

In [21]:

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print(df)
   a   b     c
0  1   2   NaN
1  5  10  20.0
```

In [22]:

#Create a DataFrame from Dict of Series

In [23]:

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
```

```
df = pd.DataFrame(d)
print(df)
   one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4
In [24]:
#Column Selection
In [25]:
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
   'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(d)
print(df)
   one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4
In [26]:
df.columns
Out[26]:
Index(['one', 'two'], dtype='object')
In [27]:
print (df ['one'])
a    1.0
b    2.0
c    3.0
d    NaN
Name: one, dtype: float64
In [28]:
print (df ['two'])
a    1
b    2
c    3
d    4
Name: two, dtype: int64
In [29]:
 #the head() methods
In [30]:
```

```
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print ("The original series is:")
print(s)

print ("The first two rows of the data series:")
print(s.head(2))
The original series is:
0    0.291512
1   -1.442382
2   -0.348753
3   -0.146003
dtype: float64
The first two rows of the data series:
0    0.291512
1   -1.442382
dtype: float64
In [31]:
#the tail() methods.
In [32]:
import pandas as pd
import numpy as np

#Create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print ("The original series is:")
print(s)

print ("The last two rows of the data series:")
print(s.tail(2))
The original series is:
0   -0.381937
1    1.209658
2   -0.624078
3    0.371913
dtype: float64
The last two rows of the data series:
2   -0.624078
3    0.371913
dtype: float64
In [33]:
```

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

#Create a DataFrame
df = pd.DataFrame(d)
print ("Our data series is:")
print(df)
Our data series is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
In [34]:
df.shape
Out[34]:
(7, 3)
In [35]:
#The transpose of the DataFrame
In [36]:
import pandas as pd
import numpy as np

# Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack']),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print ("The transpose of the data series is:")
print(df.T)
The transpose of the data series is:
           0      1      2     3      4      5     6
Name     Tom  James  Ricky   Vin  Steve  Smith  Jack
Age       25     26     25    23     30     29    23
```

```
Rating  4.23   3.24   3.98  2.56    3.2    4.6   3.8
In [37]:
#axes
In [38]:
print ("Row axis labels and column axis labels are:")
print(df.axes)
Row axis labels and column axis labels are:
[RangeIndex(start=0, stop=7, step=1), Index(['Name', 'Age', 'Rating'], dtype='object')]
In [39]:
#dtypes
In [40]:
print ("The data types of each column are:")
print(df.dtypes)
The data types of each column are:
Name       object
Age         int64
Rating    float64
dtype: object
In [41]:
#empty
In [42]:
print ("Is the object empty?")
print(df.empty)
Is the object empty?
False
In [43]:
#ndim
In [44]:
print ("Our object is:")
print (df)
print ("The dimension of the object is:")
print (df.ndim)
Our object is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The dimension of the object is:
2
In [45]:
```

```
#shape
In [46]:
print ("The shape of the object is:")
print(df.shape)
The shape of the object is:
(7, 3)
In [47]:
#size
#Number of elements in the NDFrame.
In [48]:
print ("Our object is:")
print (df)
Our object is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
In [49]:
print ("The total number of elements in our object is:")
print (df.size)
The total number of elements in our object is:
21
In [50]:
#values-Returns the actual data in the DataFrame as an NDarray.
In [51]:
print(df)
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
In [52]:
print ("The actual data in our data frame is:")
print (df.values)
The actual data in our data frame is:
[['Tom' 25 4.23]
 ['James' 26 3.24]
 ['Ricky' 25 3.98]
 ['Vin' 23 2.56]
 ['Steve' 30 3.2]
 ['Smith' 29 4.6]
 ['Jack' 23 3.8]]
```

In [53]:

#Head & Tail

#To view a small sample of a DataFrame object, use the head() and tail() methods. head() return s the first n rows (observe the index values).

In [54]:

```
print ("The first two rows of the data frame is:")
print (df.head(2))
The first two rows of the data frame is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
```

In [55]:

```
print ("The last two rows of the data frame is:")
print (df.tail(2))
The last two rows of the data frame is:
    Name  Age  Rating
5  Smith   29     4.6
6   Jack   23     3.8
```

In [56]:

#Descriptive Statistics

In [57]:

#A large number of methods collectively compute descriptive statistics and other related operati ons on DataFrame. Most of these are aggregations like sum(), mean(), but some of them, like su msum(), produce an object of the same size. Generally speaking, these methods take an axis argu ment, just like ndarray.{sum, std, ...}, but the axis can be specified by name or integer

In [58]:

#DataFrame − “index” (axis=0, default), “columns” (axis=1)

In [59]:

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack',
   'Lee','David','Gasper','Betina','Andres']),
   'Age':pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
}

#Create a DataFrame
```

```
df = pd.DataFrame(d)
print (df)
     Name  Age  Rating
0     Tom   25    4.23
1   James   26    3.24
2   Ricky   25    3.98
3     Vin   23    2.56
4   Steve   30    3.20
5   Smith   29    4.60
6    Jack   23    3.80
7     Lee   34    3.78
8   David   40    2.98
9  Gasper   30    4.80
10 Betina   51    4.10
11 Andres   46    3.65
In [60]:
```
#sum()

#Returns the sum of the values for the requested axis. By default, axis is index (axis=0

```
In [61]:
print(df.sum(0))
Name      TomJamesRickyVinSteveSmithJackLeeDavidGasperBe...
Age                                                     382
Rating                                                44.92
dtype: object
In [62]:
```
#axis=1

```
print (df.sum(1))
0     29.23
1     29.24
2     28.98
3     25.56
4     33.20
5     33.60
6     26.80
7     37.78
8     42.98
9     34.80
10    55.10
11    49.65
dtype: float64
In [63]:
```
#mean

```
print(df.mean(1))
0     14.615
```

```
1     14.620
2     14.490
3     12.780
4     16.600
5     16.800
6     13.400
7     18.890
8     21.490
9     17.400
10    27.550
11    24.825
dtype: float64
In [64]:
```
#std()

#Returns the Bressel standard deviation of the numerical columns.

```
In [65]:
print (df.std())
Age       9.232682
Rating    0.661628
dtype: float64
In [66]:
```
#Summarizing Data

#The describe() function computes a summary of statistics pertaining to the DataFrame columns.

```
In [67]:
print (df.describe())
             Age     Rating
count  12.000000  12.000000
mean   31.833333   3.743333
std     9.232682   0.661628
min    23.000000   2.560000
25%    25.000000   3.230000
50%    29.500000   3.790000
75%    35.500000   4.132500
max    51.000000   4.800000
In [68]:
```
#object − Summarizes String columns

```
print (df.describe(include=['object']))
          Name
count       12
unique      12
top     Betina
freq         1
In [69]:
```

#number − Summarizes Numeric columns

```
print (df. describe(include='number'))
            Age     Rating
count  12.000000  12.000000
mean   31.833333   3.743333
std     9.232682   0.661628
min    23.000000   2.560000
25%    25.000000   3.230000
50%    29.500000   3.790000
75%    35.500000   4.132500
max    51.000000   4.800000
In [70]:
```

#all − Summarizes all columns together (Should not pass it as a list value)

```
print (df. describe(include='all'))
         Name        Age     Rating
count       12  12.000000  12.000000
unique      12        NaN        NaN
top     Betina        NaN        NaN
freq         1        NaN        NaN
mean       NaN  31.833333   3.743333
std        NaN   9.232682   0.661628
min        NaN  23.000000   2.560000
25%        NaN  25.000000   3.230000
50%        NaN  29.500000   3.790000
75%        NaN  35.500000   4.132500
max        NaN  51.000000   4.800000
In [71]:
print(df)
      Name  Age  Rating
0      Tom   25    4.23
1    James   26    3.24
2    Ricky   25    3.98
3      Vin   23    2.56
4    Steve   30    3.20
5    Smith   29    4.60
6     Jack   23    3.80
7      Lee   34    3.78
8    David   40    2.98
9   Gasper   30    4.80
10  Betina   51    4.10
11  Andres   46    3.65
In [72]:
import pandas as pd
import numpy as np
```

#Create a Dictionary of series

```
d = {'Name':pd.Series(['Tom','James','Ricky','Vin','Steve']),
   'Age':pd.Series([25,26,25,23,30]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20])
}
```

#Create a DataFrame

```
df = pd.DataFrame(d)
print (df)
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
In [73]:
```

#reindex the DataFrame

#Reindexing changes the row labels and column labels of a DataFrame. To reindex means to co nform the data to match a given set of labels along a particular axis.

```
df_reindexed = df.reindex(index=[11,12,13,14,15], columns=['Name', 'B', 'c'])
print(df_reindexed )
   Name   B   c
11  NaN NaN NaN
12  NaN NaN NaN
13  NaN NaN NaN
14  NaN NaN NaN
15  NaN NaN NaN
In [74]:
```

#Renaming

#The rename() method allows you to relabel an axis based on some mapping (a dict or Series) or an arbitrary function.

```
In [75]:
print(df.head())
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
In [76]:
print ("After renaming the rows and columns:")
print( df.rename(columns={'Name' : 'c1', 'Age' : 'c2'},index = {0 : 'apple', 1 : 'banana', 2 : 'Mango'
}))
```

```
After renaming the rows and columns:
          c1  c2  Rating
apple     Tom  25    4.23
banana  James  26    3.24
Mango   Ricky  25    3.98
3         Vin  23    2.56
4       Steve  30    3.20
In [77]:
print(df['Name'])
0      Tom
1    James
2    Ricky
3      Vin
4    Steve
Name: Name, dtype: object
In [78]:
print(df['Age'])
0    25
1    26
2    25
3    23
4    30
Name: Age, dtype: int64
In [79]:
df.columns
Out[79]:
Index(['Name', 'Age', 'Rating'], dtype='object')
In [80]:
df.iloc[0]
Out[80]:
Name       Tom
Age         25
Rating    4.23
Name: 0, dtype: object
In [81]:
df.loc[0]
Out[81]:
Name       Tom
Age         25
Rating    4.23
Name: 0, dtype: object
In [82]:
df.head(5)
Out[82]:
       Name    Age   Rating
  0    Tom     25    4.23
  1    James   26    3.24
  2    Ricky   25    3.98
  3    Vin     23    2.56
  4    Steve   30    3.20
```

```
In [83]:
df.iloc[2][1]
Out[83]:
25
In [84]:
import numpy as np
In [85]:
 np.unique(df['Rating']).sum()
Out[85]:
17.21
In [86]:
len(np.unique(df['Rating']))
Out[86]:
5
In [87]:
df.iloc[2:3,:]
Out[87]:
       Name    Age   Rating
  2    Ricky   25    3.98
In [88]:
df.iloc[2:4,:]
Out[88]:
       Name    Age   Rating
  2    Ricky   25    3.98
  3    Vin     23    2.56
In [89]:
df.iloc[2:4,:2]
Out[89]:
     Name    Age
  2  Ricky   25
  3  Vin     23
```

```
In [90]:
y=df.iloc[:,2:]
print(y)
   Rating
0    4.23
1    3.24
2    3.98
3    2.56
4    3.20
In [91]:
x=df.iloc[:,:2]
In [92]:
x
Out[92]:
```

```
     Name    Age
  0  Tom     25
  1  James   26
  2  Ricky   25
  3  Vin     23
  4  Steve   30
```

```
In [95]:
data =pd.read_csv("D:\Datasets\EMP.csv")
In [96]:
data.head(2)
Out[96]:
```

```
     Unnamed: 0   Empid   Name   Exp   salary
  0  0            1       A      2     20000
  1  1            2       B      1     15000
```

```
In [97]:
df=pd.DataFrame(data)
In [98]:
df
Out[98]:
      Unnamed: 0   Empid   Name   Exp   salary
 0    0            1       A      2     20000
 1    1            2       B      1     15000
 2    2            3       C      15    100000
 3    3            4       D      7     80000
 4    4            5       E      20    200000
 5    5            6       F      1     15000
df.to_csv("D:\Datasets\Employee.csv")
d1=pd.read_excel("D:\Datasets\EMP.xlsx")
d1.head()
Out[103]:
      Unnamed: 0   Empid   Name   Exp   salary
 0    0            1       A      2     20000
 1    1            2       B      1     15000
 2    2            3       C      15    100000
 3    3            4       D      7     80000
 4    4            5       E      20    200000
In [104]:
d1=pd.read_excel("D:\Datasets\Emp.xlsx",sheet_name="EMP")
In [105]:
d1.head()
Out[105]:
      Unnamed: 0   Empid   Name   Exp   salary
 0    0            1       A      2     20000
 1    1            2       B      1     15000
 2    2            3       C      15    100000
 3    3            4       D      7     80000
 4    4            5       E      20    200000
In [106]:
d2=pd.read_excel("D:\Datasets\EMP.xlsx")
In [107]:
```

```
d2.head(2)
Out[107]:
```

	Unnamed: 0	Empid	Name	Exp	salary
0	0	1	A	2	20000
1	1	2	B	1	15000

```
In [108]:
d1.to_excel("D:\Datasets\E1.xlsx")
In [109]:
d2.to_excel("D:\Datasets\E12.xlsx")
In [110]:
J=df.to_json("D:\Datasets\E12.json")
In [111]:
r=pd.read_json("D:\Datasets\E12.json")
In [112]:
r.head(2)
Out[112]:
```

	Unnamed: 0	Empid	Name	Exp	salary
0	0	1	A	2	20000
1	1	2	B	1	15000

```
In [113]:
h=df.to_html("D:\Datasets\h1.html")
In [114]:
h2=pd.read_html("D:\Datasets\h1.html")
In [115]:
h2
Out[115]:
[   Unnamed: 0  Unnamed: 0.1  Empid Name  Exp  salary
 0          0             0      1    A    2   20000
 1          1             1      2    B    1   15000
 2          2             2      3    C   15  100000
 3          3             3      4    D    7   80000
 4          4             4      5    E   20  200000
 5          5             5      6    F    1   15000]
In [116]:
# Pandas Concatenation
#pd.concat(objs,axis=0,join='outer',join_axes=None,ignore_index=False)
import pandas as pd
one_df = pd.DataFrame({
   'Name': ['Raj', 'Amruta', 'Pooja', 'Sameer', 'Sanjay'],
   'subject_id':['B1','B2','B4','B6','B5'],
   'Marks_scored':[98,90,87,69,78]},
   index=[1,2,3,4,5])
one_df.head(2)
Out[119]:
        Name    subject_id    Marks_scored

  1     Raj     B1            98
```

Conclusion:

Hence we implemented Series, Data frame, Panel operations using Pandas. Also operations like Creating, Appending, Deleting. Importing different types of Datasets Also read/ write CSV, Excel, HTML,JSON file using pandas.

Assignment 6

Title: Data Visualization: Basic plotting, subplots, Histogram, box plot, bar chart, pie chart‘ line chart

Theory:

The process of finding trends and correlations in our data by representing it pictorially is called Data Visualization. To perform data visualization in python, we can use various python data visualization modules such as Matplotlib, Seaborn, Plotly, etc. Data visualization is a field in data analysis that deals with visual representation of data. It graphically plots data and is an effective way to communicate inferences from data.

Data Visualization in Python

Python offers several plotting libraries, namely Matplotlib, Seaborn and many other such data visualization packages with different features for creating informative, customized, and appealing plots to present data in the most simple and effective way.

Matplotlib and Seaborn

Matplotlib and Seaborn are python libraries that are used for data visualization. They have inbuilt modules for plotting different graphs. While Matplotlib is used to embed graphs into applications, Seaborn is primarily used for statistical graphs.

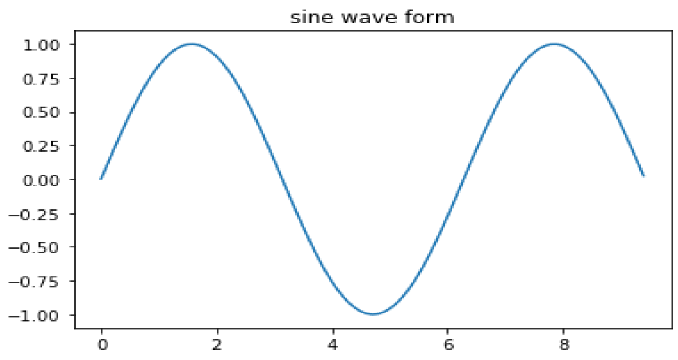
But when should we use either of the two? Let’s understand this with the help of a comparative analysis. The table below provides comparison between Python’s two well-known visualization packages Matplotlib and Seaborn.

Matplotlib	**Seaborn**
It is used for basic graph plotting like line charts, bar graphs, etc.	It is mainly used for statistics visualization and can perform complex visualizations with fewer commands.
It mainly works with datasets and arrays.	It works with entire datasets.
Seaborn is considerably more organized and functional than Matplotlib and treats the entire dataset as a solitary unit.	Matplotlib acts productively with data arrays and frames. It regards the aces and figures as objects.
Seaborn has more inbuilt themes and is mainly used for statistical analysis.	Matplotlib is more customizable and pairs well with Pandas and Numpy for Exploratory Data Analysis.

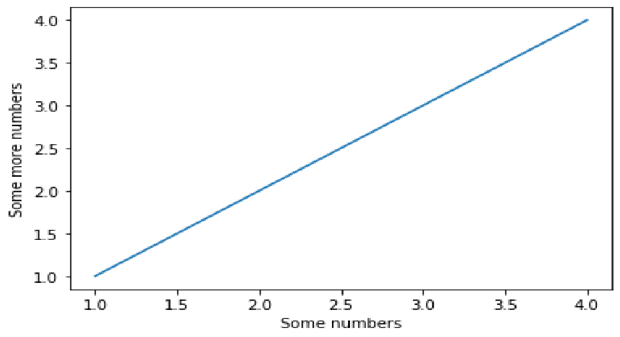
Table 1: Matplotlib vs Seaborn

Exercise:

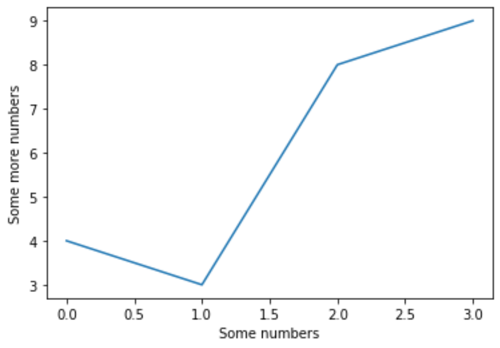
```
import numpy as np
import matplotlib.pyplot as plt
# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)
plt.title("sine wave form")
# Plot the points using matplotlib
plt.plot(x, y)
plt.show()
```



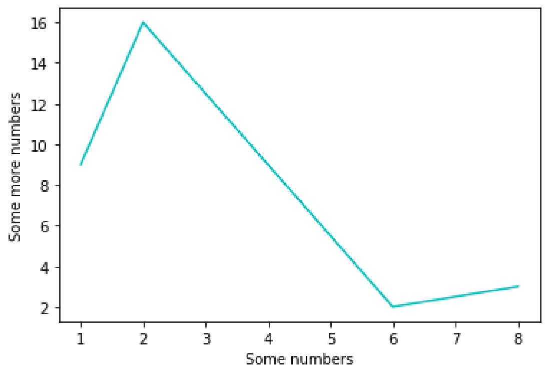
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
# Plotting a Line
plt.plot([1,2,3,4], [1,2,3,4])
plt.xlabel('Some numbers')
plt.ylabel('Some more numbers')
plt.show()
```



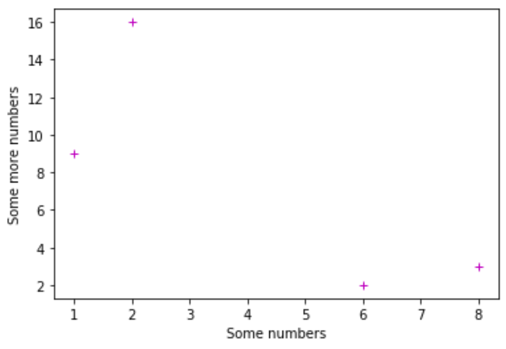
```
#Plotting using X axis only
plt.plot([4,3,8,9])
plt.xlabel('Some numbers')
plt.ylabel('Some more numbers')
plt.show()
```



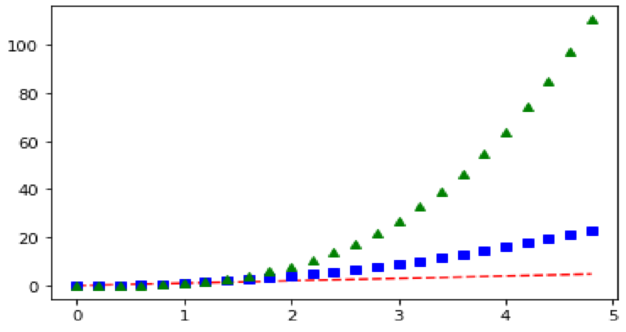
```
#Plotting with Tweaking Colors and Symbols
plt.plot([1,2,6,8], [9,16,2,3], 'c')
plt.xlabel('Some numbers')
plt.ylabel('Some more numbers')
plt.show()
```



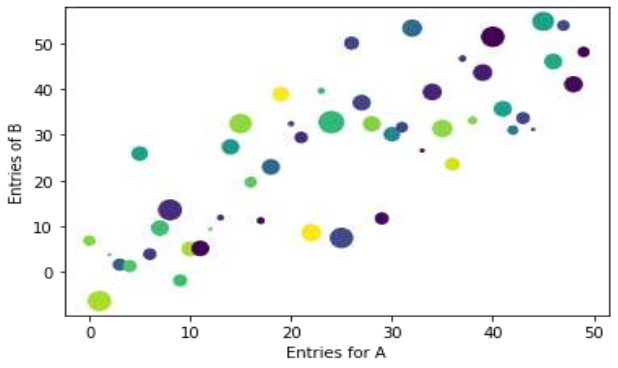
```
plt.plot([1,2,6,8], [9,16,2,3], 'm+')
plt.xlabel('Some numbers')
plt.ylabel('Some more numbers')
plt.show()
```



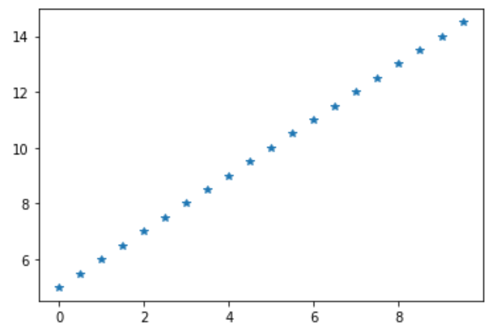
```
#Red dashes, blue squares and green triangles
t = np.arange(0., 5., 0.2)
plt.plot(t, t, 'r--', t, t**2, 'bs', t, t**3, 'g^')
plt.show()
```



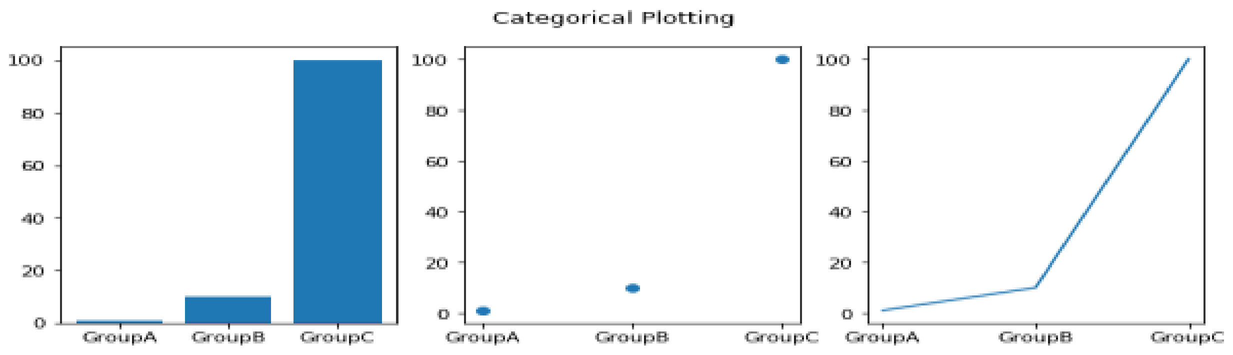
```
#By using Formula
data = np.arange(0, 10, 0.5)
y = 1 * data + 5
plt.plot(data, y, '*')
plt.show()
```



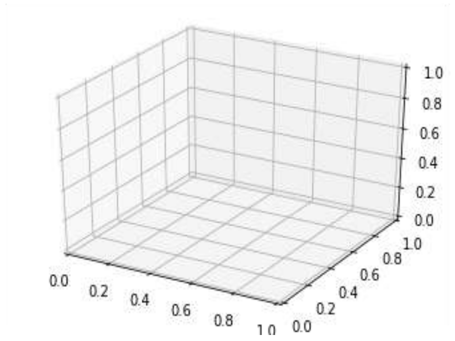
```
# By Dictionary, Color and Size
data = {
'a': np.arange(50),
'color': np.random.randint(0, 50, 50),
'size': np.random.randn(50)
}
                                   data['b'] = data['a'] + 10 * np.random.randn(50)
data['size'] = np.abs(data['size']) * 100
plt.scatter('a', 'b', c = 'color', s = 'size', data=data)
plt.xlabel('Entries for A')
plt.ylabel('Entries of B')
plt.show()
```



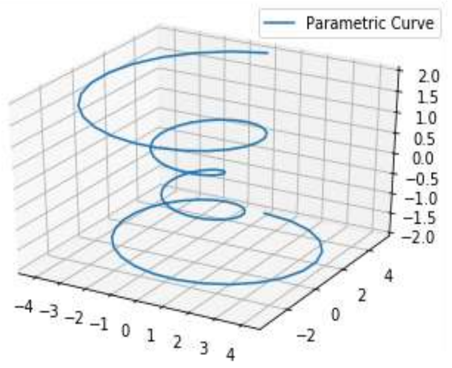
```
#Plotting with Categorical Data
names = ['GroupA', 'GroupB', 'GroupC']
values = [1, 10, 100]
plt.figure(1, figsize=(10,4))
plt.subplot(131)
plt.bar(names, values)
plt.subplot(132)
plt.scatter(names,values)
plt.subplot(133)
plt.plot(names, values)
plt.suptitle('Categorical Plotting')
plt.show()
```



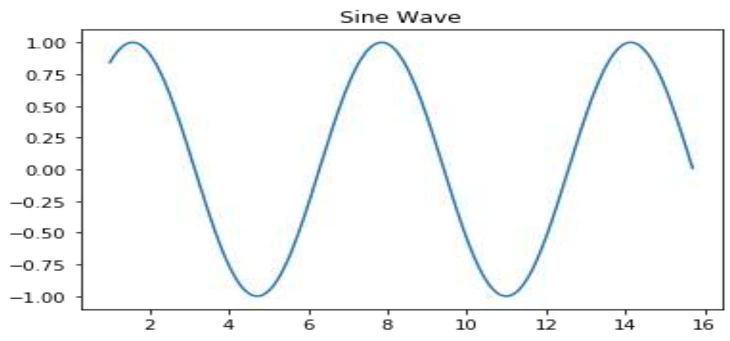
```
#Plotting over 3D Axes
import matplotlib as mpl
from mpl_toolkits.mplot3d import Axes3D
mpl.rcParams['legend.fontsize'] = 10
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

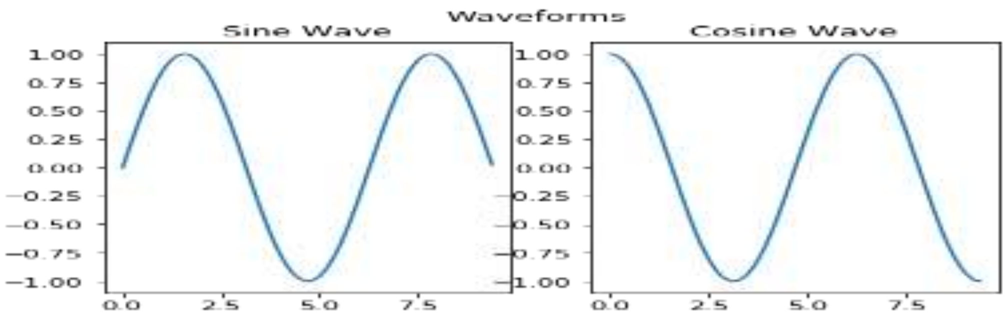


```
#Parametric Curve
mpl.rcParams['legend.fontsize'] = 10
fig = plt.figure()
ax = fig.gca(projection='3d')
theta = np.linspace(-4 * np.pi, 4 * np.pi, 100)
z = np.linspace(-2, 2, 100)
r = z**2 + 1
x = r * np.sin(theta)
y = r * np.cos(theta)
ax.plot(x, y, z, label='Parametric Curve')
ax.legend()
plt.show()
```

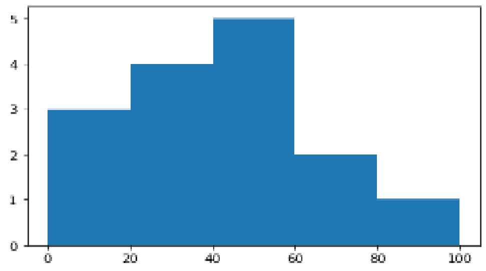


```
#Subplots of Sine and Cosine
x = np.arange(1, 5 * np.pi, 0.01)
y = np.sin(x)
plt.title('Sine Wave')
plt.plot(x, y)
plt.show()

x = np.arange(0, 3* np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)
plt.subplot(1, 2, 1)
plt.plot(x, y_sin)
plt.title('Sine Wave')
plt.subplot(1, 2, 2)
plt.plot(x, y_cos)
plt.title('Cosine Wave')
plt.suptitle('Waveforms')
plt.show()
```

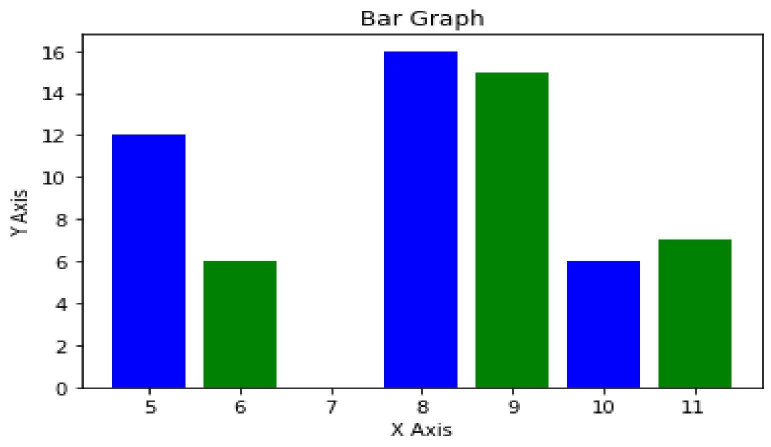




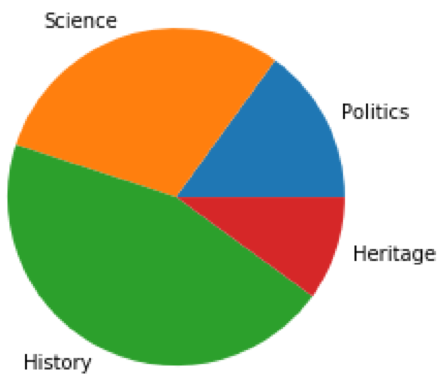
```
#Plotting of Histogram
a = np.array([22, 87, 5, 43, 56, 73, 55, 54, 11, 20, 51, 5, 79, 31, 27])
plt.hist(a, bins=[0, 20, 40, 60, 80, 100])
plt.show()
```



```
#Plotting using BAR Graph
x1 = [5, 8, 10]
y1 = [12, 16, 6]
x2 = [6, 9, 11]
y2 = [6, 15, 7]
plt.bar(x1, y1, color = 'b')
plt.bar(x2, y2, color = 'g', align='center')
plt.title('Bar Graph')
plt.ylabel('Y Axis')
plt.xlabel('X Axis')
plt.show()
```



```
#PIE Chart Plotting
labels = ['Politics', 'Science', 'History', 'Heritage']
interest = [15, 30, 45, 10]
fig1, ax1 = plt.subplots()
ax1.pie(interest, labels=labels)
plt.show()
```



```
#Bar Chart Plotting
import matplotlib.pyplot as plt
# x-coordinates of left sides of bars
left = [1, 2, 3, 4, 5]
# heights of bars
height = [10, 24, 36, 40, 5]
# labels for bars
tick_label = ['one', 'two', 'three', 'four', 'five']
```

```
# plotting a bar chart
plt.bar(left, height, tick_label = tick_label,
 width = 0.8, color = ['red', 'green'])
# naming the x-axis
plt.xlabel('x - axis')
# naming the y-axis
plt.ylabel('y - axis')
# plot title
plt.title('My bar chart!')
# function to show the plot
plt.show()
```

Conclusion:

We studied the concept of Data Visualization and visualized the data in various form like subplots, Histogram, box plot, bar chart, pie chart, line chart.

Assignment 7

Title: Data Preprocessing/Wrangling: Basic data handling, reshaping, pivoting, rank and sort data, basic datagrouping

Theory:

Data wrangling is one of the crucial tasks in data science and analysis which includes operations like:

- **Data Sorting:** To rearrange values in ascending or descending order.
- **Data Filtration:** To create a subset of available data.
- **Data Reduction:** To eliminate or replace unwanted values.
- **Data Access**: To read or write data files.
- **Data Processing:** To perform aggregation, statistical, and similar operations on specific values.

Pandas is an open source library, specifically developed for data science and analysis. It is built upon the Numpy (to handle numeric data in tabular form) package and has inbuilt data structures to ease-up the processof data manipulation, aka data munging/wrangling.

Exercise:

Pivot Table

```
# Create a simple dataframe
# importing pandas as pd
import pandas as pd
import numpy as np

# creating a dataframe
df = pd.DataFrame({'A': ['John', 'Boby', 'Mina', 'Peter', 'Nicky'],
 'B': ['Masters', 'Graduate', 'Graduate', 'Masters', 'Graduate'],
 'C': [27, 23, 21, 23, 24]})
 df
```

	A	B	C
0	John	Masters	27
1	Boby	Graduate	23
2	Mina	Graduate	21
3	Peter	Masters	23
4	Nicky	Graduate	24

```
# Simplest pivot table must have a dataframe
# and an index/list of index.
table = pd.pivot_table(df, index =['A', 'B'])
table
```

A	B	c
Boby	**Graduate**	23
John	**Masters**	27
Mina	**Graduate**	21
Nicky	**Graduate**	24
Peter	**Masters**	23

```
# Creates a pivot table dataframe
table = pd.pivot_table(df, values ='A', index =['B', 'C'],columns =['B'], aggfunc = np.sum)
table
```

	B	Graduate	Masters
B	C		
Graduate	**21**	Mina	NaN
	23	Boby	NaN
	24	Nicky	NaN
Masters	**23**	NaN	Peter
	27	NaN	John

Melt Function

```
# Create a simple dataframe

# importing pandas as pd
import pandas as pd

# creating a dataframe
df = pd.DataFrame({'Name': {0: 'John', 1: 'Bob', 2: 'Shiela'},
 'Course': {0: 'Masters', 1: 'Graduate', 2: 'Graduate'},
 'Age': {0: 27, 1: 23, 2: 21}})
df
```

	Name	Course	Age
0	John	Masters	27

	Name	Course	Age
1	Bob	Graduate	23
2	Shiela	Graduate	21

```
# Name is id_vars and Course is value_vars
pd.melt(df, id_vars =['Name'], value_vars =['Course'])
```

	Name	variable	value
0	John	Course	Masters
1	Bob	Course	Graduate
2	Shiela	Course	Graduate

```
# multiple unpivot columns
pd.melt(df, id_vars =['Name'], value_vars =['Course', 'Age'])
```

	Name	variable	value
0	John	Course	Masters
1	Bob	Course	Graduate
2	Shiela	Course	Graduate
3	John	Age	27
4	Bob	Age	23
5	Shiela	Age	21

```
# Names of _variable‘ and _value‘ columns can be customized
pd.melt(df, id_vars =['Name'], value_vars =['Course'],
var_name ='ChangedVarname', value_name ='ChangedValname')
```

	Name	ChangedVarname	ChangedValname
0	John	Course	Masters
1	Bob	Course	Graduate
2	Shiela	Course	Graduate

<u>Sort Function</u>

```
#importing pandas package
import pandas as pd
#making data frame from csv file
data=pd.read_csv("C:\\Users\\Administrator\\Desktop\\nba.csv")
#sorting data frame by Team and then By names
data.sort_values(["Team", "Name"], axis=0,
ascending=True, inplace=True)
#display
data
```

Output : Display all the data of the CSV file.

458 rows × 9 columns

```
import pandas as pd
 #making data frame from csv file
data=pd.read_csv("nba.csv")
 #sorting data frame by Team and then By names
data.sort_values(["Team", "Name"], axis=0,
 ascending=[True,False], inplace=True)
data
```

Output : Display all the data of 458 rows × 9 columns from CSV file in sorted format according to Team & Name.

Out[9]:

```
#importing pandas package
import pandas as pd
data=pd.read_csv("nba.csv")
 #sorting data frame by Team, age and height
data.sort_values(["Team", "Age", "Height"], axis=0,
 ascending=[False,True,False],
inplace=True)
data
```

Output: Displayed the contents of *data frame by Team, age and height*

```
import pandas as pd
data = pd.read_csv("nba.csv")
 # sorting w.r.t team name
data.sort_values("Team", inplace = True)
 # creating a rank column and passing the returned rank series
# change method to 'min' to rank by minimum
```

```
data["Rank"] = data["Team"].rank(method ='average')
 data
```

Output: Displayed the contents of *data frame*

```
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.read_csv("nba.csv")

# Print the dataframe
df
```

Output: 458 rows × 9 columns

```
# applying groupby() function to
# group the data on team value.
gk = df.groupby('Team')

# Let's print the first entries in all the groups formed.
gk.first()
```

Output: It displayed *the first entries in all the groups.*:

```
# Finding the values contained in the "Boston Celtics" group
gk.get_group('Boston Celtics')
# importing pandas as pd
import pandas as pd

# Creating the dataframe
df = pd.read_csv("nba.csv")

# First grouping based on "Team"
# Within each team we are grouping based on "Position"
gkk = df.groupby(['Team', 'Position'])

# Print the first value in each group
gkk.first()
```

Output: It displayed first value of *"Boston Celtics" group* **by making group of team**

```
# importing pandas as pd
import pandas as pd
 # Creating the dataframe
df = pd.read_csv("nba.csv")

# First grouping based on "Team"
# Within each team we are grouping based on "Position"
gkk = df.groupby(['Team', 'Position'])
 # Print the first value in each group
gkk.first()
```

Output: It displayed first value of each team.

Conclusion:

We have performed data preprocessing/ wrangling by executing various operations such as data handling, reshaping, pivoting, rank and sort data, basic datagrouping.

ASSIGNMENT NUMBER 8

Title: Data Preprocessing/Wrangling: ETL Phase 1 and Phase 2

Theory:

Data Preprocessing

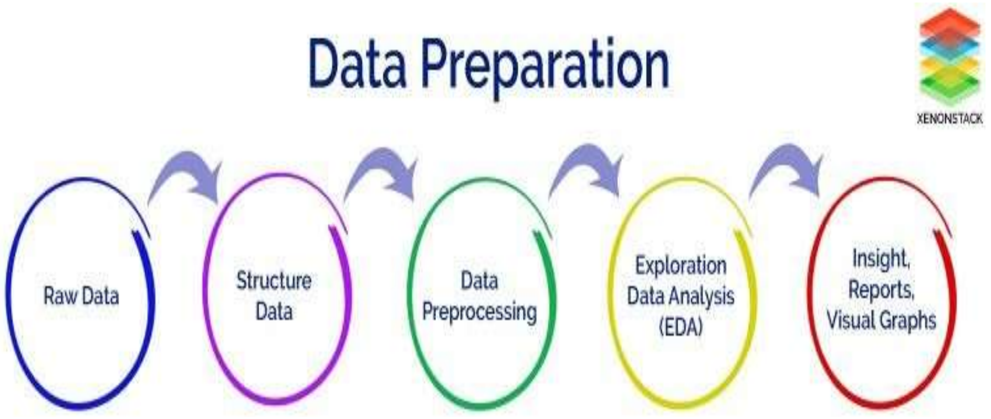
Deep learning and Machine learning are becoming more and more important in today‘s ERP (Enterprise Resource Planning). During the process of building the analytical model using Deep Learning or MachineLearning the data set is collected from various sources such as a file, database, sensors and much more.

But, the collected data cannot be used directly for performing analysis process. Therefore, to solv e thisproblem **Data Preparation** is done. It includes two techniques that are listed below

- Data Preprocessing
- **Data Wrangling**

Data Preprocessing Architecture:

Data Preparation is an important part of Data Science. It includes two concepts such as **Data Cle aning** and**Feature Engineering**. These two are compulsory for achieving better accuracy and pe rformance in the Machine Learning and Deep Learning projects



Data Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for theanalysis.

Therefore, certain steps are executed to convert the data into a small clean data set. This technique is performed before the execution of **Iterative Analysis**. The set of steps is known as Data Preprocessing. Itincludes –

- Data Cleaning
- Data Integration
- Data Transformation
- Data Reduction

Need of Data Preprocessing

For achieving better results from the applied model in Machine Learning and Deep Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning and Deep Learning modelneed information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values has to be managed from the original raw data set.

Another aspect is that the data set should be formatted in such a way that more than one Machine Learningand Deep Learning algorithms are executed in one data set, and best out of them is chosen.

What is Data Wrangling?

Data Wrangling is a technique that is executed at the time of making an interactive model. In other words, itis used to convert the raw data into the format that is convenient for the consumption of data.

This technique is also known as **Data Munging**. This method also follows certain steps such as afterextracting the data from different data sources, sorting of data using certain algorithm is performed, decompose the data into a different structured format and finally store the data into another database.

Need of Data Wrangling

Data Wrangling is an important aspect of implementing the model. Therefore, data is converted to the properfeasible format before applying any model to it. By performing filtering, grouping and selecting appropriate data accuracy and performance of the model could be increased.

Another concept is that when time series data has to be handled every algorithm is executed with different aspects. Therefore, Data Wrangling is used to convert the time series data into the required format of the applied model. In simple words, the complex data is transformed into a usable format for performing analysison it.

Why is Data Preprocessing Important?

Data Preprocessing is necessary because of the presence of unformatted real-world data. Mostly real-worlddata is composed of –

- **Inaccurate data (missing data) –** There are many reasons for missing data such as data is not continuously collected, a mistake in data entry, technical problems with biometrics and much more.

- **The presence of noisy data (erroneous data and outliers) –** The reasons for the existence of noisydata could be a technological problem of gadget that gathers data, a human mistake during data entryand much more.

- **Inconsistent data –** The presence of inconsistencies are due to the reasons such that existence of duplication within data, human data entry, containing mistakes in codes or names, i.e., violation ofdata constraints and much more.

Therefore, to handle raw data, Data Preprocessing is performed.

Why is Data Wrangling Important?

Data Wrangling is used to handle the issue of **Data Leakage** while implementing

Machine Learning andDeep Learning. First of all, we have to understand what Data Leakage is?

Data Leakage in Machine Learning and Deep Learning

Data Leakage is responsible for the cause of invalid Machine Learning/Deep Learning model due to the overoptimization of the applied model.

Data Leakage is the term used when the data from outside, i.e., not part of training dataset is used for the learning process of the model. This additional learning of information by the applied model will disapprovethe computed estimated performance of the model.

For example when we want to use the particular feature for performing **Predictive Analysis**, but that specificfeature is not present at the time of training of dataset then data leakage will be introduced within the model.

Data Leakage can be demonstrated in many ways that are given below –

- The Leakage of data from test dataset to training data set.
- Leakage of computed correct prediction to the training dataset.
- Leakage of future data into the past data.
- Usage of data outside the scope of the applied algorithm

In general, the leakage of data is observed from two primary sources of Machine Learning/Deep Learnin galgorithms such as feature attributes (variables) and training data set.

Exercise:

```
from sklearn import preprocessing
import pandas as pd
housing = pd.read_csv("C:\\Users\\Administrator\\Desktop\\california_housing_train.csv")
scaler = preprocessing.MinMaxScaler()
names = housing.columns
d = scaler.fit_transform(housing)
scaled_df = pd.DataFrame(d, columns=names)
scaled_df.head()
```

Out[4]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	1.000000	0.175345	0.274510	0.147885	0.198945	0.028364	0.077454	0.068530	0.107012
1	0.984064	0.197662	0.352941	0.201608	0.294848	0.031559	0.075974	0.091040	0.134228
2	0.975100	0.122210	0.313725	0.018927	0.026847	0.009249	0.019076	0.079378	0.145775
3	0.974104	0.116897	0.254902	0.039515	0.052142	0.014350	0.037000	0.185639	0.120414

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
4	0.974104	0.109458	0.372549	0.038276	0.050435	0.017405	0.042921	0.098281	0.104125

```
import numpy as np
x_array = np.array([2,3,5,6,7,4,8,7,6])
normalized_arr = preprocessing.normalize([x_array])
print(normalized_arr)

[[0.11785113 0.1767767  0.29462783 0.35355339 0.41247896 0.23570226
  0.47140452 0.41247896 0.35355339]]
from sklearn import preprocessing
import pandas as pd
housing = pd.read_csv("C:\\Users\\Administrator\\Desktop\\california_housing_train.csv")
d = preprocessing.normalize(housing)
scaled_df = pd.DataFrame(d, columns=names)
scaled_df.head()
```

Out[6]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	-0.001702	0.000509	0.000223	0.083566	0.019105	0.015114	0.007028	0.000022	0.996178
1	-0.001422	0.000427	0.000236	0.095035	0.023616	0.014025	0.005752	0.000023	0.995077
2	-0.001337	0.000393	0.000198	0.008401	0.002030	0.003885	0.001365	0.000019	0.999953
3	-0.001561	0.000458	0.000191	0.020444	0.004590	0.007015	0.003078	0.000043	0.999750
4	-0.001749	0.000512	0.000305	0.022192	0.004976	0.009524	0.003999	0.000029	0.999686

```
from sklearn import preprocessing
import pandas as pd
housing = pd.read_csv("C:\\Users\\Administrator\\Desktop\\california_housing_train.csv")
scaler = preprocessing.MinMaxScaler(feature_range=(0, 2))
```

```
names = housing.columns
d = scaler.fit_transform(housing)
scaled_df = pd.DataFrame(d, columns=names)
scaled_df.head()
```

Out[8]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
0	2.000000	0.350691	0.549020	0.295769	0.397890	0.056728	0.154909	0.137060	0.214024
1	1.968127	0.395324	0.705882	0.403216	0.589696	0.063118	0.151949	0.182080	0.268457
2	1.950199	0.244421	0.627451	0.037854	0.053693	0.018498	0.038152	0.158756	0.291549
3	1.948207	0.233794	0.509804	0.079030	0.104283	0.028700	0.074001	0.371278	0.240828
4	1.948207	0.218916	0.745098	0.076552	0.100869	0.034810	0.085841	0.196563	0.208251

```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(housing, test_size=0.2, random_state=42)
test_set.head()
```

Out[10]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
10941	-120.87	37.77	9.0	4838.0	920.0	2460.0	923.0	3.5959	142700.0
5250	-118.14	34.11	52.0	2742.0	422.0	1153.0	414.0	8.1124	500001.0
10292	-120.05	36.98	16.0	3705.0	739.0	2463.0	697.0	2.5288	61800.0

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
2266	-117.42	34.02	9.0	5455.0	882.0	3015.0	858.0	4.2321	162800.0
6398	-118.26	33.97	52.0	1331.0	346.0	1144.0	362.0	1.5326	90600.0

```
from sklearn.preprocessing import OneHotEncoder

housing_cat = housing[["households"]]
cat_encoder = OneHotEncoder(sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(housing_cat)
housing_cat_1hot
```

Out[12]:

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

Conclusion:
We have executed data pre-processing in ETL in phase-I and phase-II by importing california_ housing train csv file. Also studied the concept of Data wrangling.

Assignment No 9

Title: Implementing Web Scraping in Python

Theory:
There are mainly two ways to extract data from a website:
- Use the API of the website (if it exists). For example, Facebook has the Facebook Graph API which allows retrieval of data posted on Facebook.
- Access the HTML of the webpage and extract useful information/data from it. This technique is called web scraping or web harvesting or web data extraction.

This article discusses the steps involved in web scraping using implementation of Web Scraping in Python with Beautiful Soup
Steps involved in web scraping:
1. Send a HTTP request to the URL of the webpage you want to access . The server responds to the request by returning the HTML content of the webpage. For this task , we will use a third-party HTTP library for python requests.
2. Once we have accessed the HTML content, we are left with the task of parsing the data. Since most of the HTML data is nested, we cannot extract data simply throug h string processing. One needs a parser which can create a nested/tree structure of the HTML dat a.

There are many HTML parser libraries available but the most advanced one is html5lib.
Now, all we need to do is navigating and searching the parse tree that we created, i.e. tree travers al. For this task, we will be using another third-party python library, Beautiful Soup. It is a Pytho n library for pulling data out of HTML and XML files.
Step 1: Installing the required third-party libraries
- Easiest way to install external libraries in python is to use pip. **pip** is a packag e management system used to install and manage software packages written in Python.

All you need to do is: pip install requests
pip install html5lib pip install bs4
- Another way is to download them manually from these links:
- requests
- html5lib

beautifulsoup4

Exercise:
```
# import libraries
from bs4 import BeautifulSoup
import pandas as pd
import csv
import requests
import bs4 as bs
# specify the url
url= 'https://riddhinilawar.wordpress.com/'
page= requests.get(url)
page
```

Out[3]:

```
<Response [200]>
```

In [4]:

```
page.content
```

Out[4]:

```
b'<!doctype html>\n<html lang="en">\n<head>\n\t<meta charset="UTF-8" />\n\t<meta name="
viewport" content="width=device-width, initial-scale=1" />\n\t<link rel="profile" href="https://g
mpg.org/xfn/11" />\n\t<title>Indeed voice</title>\n<meta name=\'robots\' content=\'max-image-
preview:large\' />\n<link rel=\'dns-prefetch\' href=\'//s2.wp.com\' />\n<link rel=\'dns-prefetch\' hr
ef=\'//s1.wp.com\' />\n<link rel=\'dns-prefetch\' href=\'//s0.wp.com\' />\n<link rel=\'dns-prefetch\'
href=\'//wordpress.com\' />\n<link rel=\'dns-prefetch\' href=\'//fonts.googleapis.com\' />\n<link re
l=\'dns-prefetch\' href=\'//s.pubmine.com\' />\n<link rel=\'dns-prefetch\' href=\'//x.bidswitch.net\'
/>\n<link rel=\'dns-prefetch\' href=\'//static.criteo.net\' />\n<link rel=\'dns-prefetch\' href=\'//ib.ad
nxs.com\' />\n<link rel=\'dns-prefetch\' href=\'//aax.amazon-adsystem.com\' />\n<link rel=\'dns-p
refetch\' href=\'//bidder.criteo.com\' />\n<link rel=\'dns-prefetch\' href=\'//cas.criteo.com\' />\n<li
nk rel=\'dns-prefetch\' href=\'//gum.criteo.com\' />\n<link rel=\'dns-prefetch\' href=\'//ads.pubmat
ic.com\' />\n<link rel=\'dns-prefetch\' href=\'//gads.pubmatic.com\' />\n<link rel=\'dns-prefetch\'
href=\'//tpc.googlesyndication.com\' />\n<link rel=\'dns-prefetch\' href=\'//ad.doubleclick.net\' />\
n<link rel=\'dns-prefetch\' href=\'//googleads.g.doubleclick.net\' />\n<link rel=\'dns-prefetch\' hre
f=\'//www.googletagservices.com\' />\n<link rel=\'dns-prefetch\' href=\'//cdn.switchadhub.com\' /
>\n<link rel=\'dns-prefetch\' href=\'//delivery.g.switchadhub.com\' />\n<link rel=\'dns-prefetch\' h
ref=\'//delivery.swid.switchadhub
```

Conclusion:
We have performed web scrapping by importing beautiful soap tool.

Assignment No. 10

Title: Data transformation and Exploratory Data Analysis

Theory:

Exploratory data analysis (EDA) is a crucial component of data science which allows you to develop the gist of what your data look like and what kinds of questions might be answered by them. Ultimately, EDA is important because it allows the investigator to make critical decisions about what is interesting to pursue and what probably isn‘t worth following up on and thus building a hypothesis using the relationships between variables.

There are various methods used to transform variables. As discussed, some of them include square root, cube root, logarithmic, binning, reciprocal and many others. Let‘s look at these methods in detail by highlighting the pros and cons of these transformation methods.

- **Logarithm:** Log of a variable is a common transformation method used to change the shape of distribution of the variable on a distribution plot. It is generally used for reducing right skewness of variables. Though, It can‘t be applied to zero or negative values as well.

- **Square / Cube root:** The square and cube root of a variable has a sound effect on variable distribution. However, it is not as significant as logarithmic transformation. Cuberoot has its own advantage. It can be applied to negative values including zero. Square root can be applied to positive values including zero.

- **Binning:** It is used to categorize variables. It is performed on original values, percentileor frequency. Decision of categorization technique is based on business understanding. For example, we can categorize income in three categories, namely: High, Average and Low. We can also perform co-variate binning which depends on the value of more than one variable

Exercise:

```
import pandas as pd
#Import Library Pandas
df = pd.read_csv("C:\\Users\\Administrator\\Desktop\\train.csv") #I am working in Windows envi
ronment
#Reading the dataset in a dataframe using Pandas
print (df.head(3))#Print first three observations
```

```
  PassengerId  Survived  Pclass  \
0           1         0       3
1           2         1       1
2           3         1       3
```

```
                                               Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
```

```
df=pd.read_excel("C:\\Users\\Administrator\\Desktop\\sample-xls-file-for-testing.xls")
print (df.head(3))
```

```
      Segment  Country    Product Discount Band  Units Sold  \
0  Government   Canada  Carretera          None      1618.5
1  Government  Germany  Carretera          None      1321.0
2   Midmarket   France  Carretera          None      2178.0

   Manufacturing Price  Sale Price  Gross Sales  Discounts    Sales     COGS  \
0                    3          20      32370.0        0.0  32370.0  16185.0
1                    3          20      26420.0        0.0  26420.0  13210.0
2                    3          15      32670.0        0.0  32670.0  21780.0

    Profit       Date  Month Number Month Name  Year
0  16185.0 2014-01-01             1    January  2014
1  13210.0 2014-01-01             1    January  2014
2  10890.0 2014-06-01             6       June  2014
```

```
df=pd.read_csv("C:\\Users\\Administrator\\Desktop\\s2.txt") # Load Data from text file having ta
b ‗\t‘ delimeter print
print (df.head(6))
```

```
  Time (h)\tMannitol\tInositol\tSorbitol\tRhamnose\tSucrose\tLactose
0   1\t 0.2054183\t 0.1266039\t 0.2485959\t 0.779...
1   2\t 0.4548433\t 0.1653758\t 0.5369149\t 0.910...
2   3\t 1.1309680\t 0.1948297\t 0.5841197\t 1.224...
3   4\t 2.0990684\t 0.3243677\t 0.6332420\t 1.205...
4   5\t 3.0617158\t 0.5317284\t 1.1851830\t 1.681...
5   6\t 3.7896505\t 1.0598055\t 2.3468473\t 3.078...
```

```
df=pd.read_excel("C:\\Users\\Administrator\\Desktop\\salary.xlsx") # Load Data sheet of excel fi
le EMP
print (df)
print(df.describe())
#result=df.pivot(index= 'id', columns='Status', values='Salary')
```

```
#print(result)
   Employee List  Unnamed: 1 Unnamed: 2 Unnamed: 3
0           NaN        NaN       NaN       NaN
1      Last Name  First Name     Status     Salary
2       Williams       Mary  Full Time      35000
3         Brown     Robert  Full Time      32000
4        Wilson   Elizabeth  Part Time      12000
5         Moore    Jennifer  Full Time      41000
6         Brown     Charles  Full Time      39000
7         Price       Lisa  Part Time      14000
8          Wood      Daniel  Part Time      13750
9       Coleman      Donald  Full Time      37500
10        Perry      George  Part Time      12050
11       Steele       Donna  Full Time      36750
12      Schultz       Carol  Full Time      38050
13        Munoz        Ruth  Part Time      11000
14     Chandler       Jason  Full Time      29000
15        Small     Matthew  Full Time      45500
16      Hensley     Jessica  Full Time      52000
17        Brown        Gary  Part Time       8000
18       Grimes        Jose  Part Time      17000
19       Baxter      Brenda  Full Time      36000
20        Morin       Frank  Full Time      36500
21      Tillman    Kathleen  Part Time       9750
22        Huber      Joshua  Full Time      31750
23        Boyle       Debra  Full Time      38050
24      Buckner       Jerry  Full Time      37500
25      Knowles       Aaron  Part Time      10050
26    Velazquez      Carlos  Part Time       9075
27         Vang     Marilyn  Full Time      29750
       Employee List Unnamed: 1 Unnamed: 2  Unnamed: 3
count            27         27         27          27
unique           25         27          3          25
top           Brown      Frank  Full Time       38050
freq              3          1         16           2

df=pd.read_excel("C:\\Users\\Administrator\\Desktop\\EXAMPLE.xlsx") # Load Data sheet of e
xcel file EMP
print (df)
print(df.describe())
result= df.pivot(index= 'ID', columns='PRODUCT', values='SALES')
result
```

```
  ID PRODUCT  SALES      EMP
0   1      AA     43      JOE
1   2      AB     45     JACK
2   3      AC     21     JAZZ
3   4      AD     34     JOHN
4   5      BA     56     MARY
5   6      BB     55   MICHAL
6   7      BC     36     ROSS
7   8      BD     48  ADMINISTRATOR
8   9      CA     50      LEO
9  10      CC     52    SHANU
             ID      SALES
count  10.00000  10.000000
mean    5.50000  44.000000
std     3.02765  10.934146
min     1.00000  21.000000
25%     3.25000  37.750000
50%     5.50000  46.500000
75%     7.75000  51.500000
max    10.00000  56.000000
```

Out[47]:

PRODUCT	AA	AB	AC	AD	BA	BB	BC	BD	CA	CC
ID										
1	43.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	NaN	45.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	21.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	34.0	NaN	NaN	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	56.0	NaN	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	NaN	55.0	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	NaN	NaN	36.0	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	48.0	NaN	NaN
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	50.0	NaN
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	52.0

```
#Sorting Dataframe
df1=pd.read_excel("C:\\Users\\Administrator\\Desktop\\EXAMPLE.xlsx", "Sheet1") #Add by va
riable name(s) to sort
df1.sort_values(['PRODUCT','SALES'], ascending=[True, False])
```

```
                                        Out[54]:
     ID  PRODUCT  SALES            EMP
0     1       AA     43            JOE
1     2       AB     45           JACK
2     3       AC     21           JAZZ
3     4       AD     34           JOHN
4     5       BA     56           MARY
5     6       BB     55         MICHAL
6     7       BC     36           ROSS
7     8       BD     48  ADMINISTRATOR
8     9       CA     50            LEO
9    10       CC     52          SHANU
```

Conclusion:
We have performed various operations like sort, Indexing, sorting dataframe. Also performed Various opration such as finding head, columns of dataframe. Also read the excel file in python.

Content Beyond Syllabus

Assignment No. 11

Title: Perform the following operations using R/Python on the sample data set

a) Create data subsets
b) Merge data
c) Sort data
d) Transposing data
e) Melting Data to long format
f) Casting Data to wide format

Learning Objective:

Data Exploration- Cleaning , Data Processing

THEORY:

- Pre-processing refers to the transformations applied to our data before feeding it to thealgorithm.
- Data Preprocessing is a technique that is used to convert the raw data into a clean data set. Inother words, whenever the data is gathered from different sources it is collected in raw formatwhich is not feasible for the analysis.

Exerci **EXCERCISE:**

```
getwd()

setwd("C://Users//RGI//Desktop//dsbda1

")print(getwd())

data <-

read.csv("facebook.csv")

print(data)

head(data)

print(is.data.frame(d

ata))

print(ncol(data))

print(nrow(data))
print("****************Create data
subset**************")sub=data[c('userid','age','likes')]
head(sub)
write.csv(sub,"facebook-
sub.csv")
subset1=subset(sub,age>16)
subset1
head(subset1)
print("*************Merge
data***************")data1 <-
read.csv("facebook.csv")
data2 <-
read.csv("facebook.csv")
head(data2)
dim(
data1
)
dim(
data2
)
newCD=rbind(data1,data2)
dim(newCD)
```

```
print("****************Sort data***********")sub

x=sub[order(-data$age),]head(x)

y=sub[order(-data$likes),]head(y)


print("*****************Reshape data in wide format************")

install.packages("reshape")

library(reshape)sub

head(melt(data = sub,id.vars = "userid"))sub

sub=data[c('Post.userid','Post.userage','likes')]sub

head(sub) cast(sub,Post.userid~Postuserage.,value = '1')
```

<u>OutPut:</u>

```
29 1670750   15     0
30 1041376   15     0
31 1762274   15     0
32 1903650   14     0
33 2087235   14     0
34 1871735   14     0
35 1459785   14     0
36 1215208   14     0
37 1255528   14     0
38 1390333   14     0
39 1570689   14     0
```

```
40 1197377   14     0
41 1472999   14     0
42 1549614   14     0
43 2056802   14     0
44 2091823   14     0
45 1302193   14     0
46 1784528   14     0
47 1842851   14     0
48 1002019   14     0
49 2188166   14     0
50 1443541   14     0

[ reached 'max' / getOption("max.print") -- omitted 98670 rows ]
> head(melt(data = sub,id.vars = "userid"))userid
variable value
1 2094382      age    14
2 1192601      age    14
3 2083884      age    14
4 1203168      age    14
5 1733186      age    14
6 1524765      age    14
sub
userid age likes
1  2094382   14     0
2  1192601   14     0
3  2083884   14     0
4  1203168   14     0
5  1733186   14     0
6  1524765   14     0
7  1136133   13     0
8  1680361   13     0
9  1365174   13     0
10  1712567 13     0
11  1612453 13     0
12  2104073 13     0
13  1918584 13     0
14  1704433 13     0
15  1932519 13     0
16  1751722 13     0
17  1470850 13     0
18  1001768 13     0
19  1537661 13     0
20  1020296 13     0
21  1472643 13     0
22  2041297 13     0
23  1514978 13     0
24  1708962 15     0
25  1098955 15     0
```

```
26 1001243   15      0
27 2113084   15      0
28 2163454   15      0
29 1670750   15      0
30 1041376   15      0
31 1762274   15      0
32 1903650   14      0
33 2087235   14      0
34 1871735   14      0
35 1459785   14      0
36 1215208   14      0
37 1255528   14      0
38 1390333   14      0
39 1570689   14      0
40 1197377   14      0
41 1472999   14      0
42 1549614   14      0
43 2056802   14      0
44 2091823   14      0
45 1302193   14      0
46 1784528   14      0
47 1842851   14      0
48 1002019   14      0
49 2188166   14      0
50 1443541   14      0
51 1927282   14      0
52 1929885   14      0
53 2188851   14      0
54 1562606   14      0
55 1599485   14      0
56 2157419   14      0
57 1264260   14      0
58 1123633   14      0
59 2045259   14      0
60 1712642   14      0
61 1304645   14      0
62 1287843   14      0
63 1094094   14      0
64 1201176   14      0
65 1717785   14      0
66 1764393   14      0
67 1483573   14      0
68 2163813   16      0
69 1435172   16      0
70 1732320   16      0
71 1866176   16      0
72 1402504   16      0
73 1469691   16      0
```

```
             74 1820036   15     0
             75 1922550   15     0
             76 1450870   15     0
             77 2044883   15     0
             78 2055501   15     0
             79 1574525   15     0
             80 1460963   15     0
             81 1648391   15     0
             82 2162779   15     0
             83 1105499   15     0
             84 1654852   15     0
             85 1665635   15     0
             86 2113685   15     0
             87 1772186   15     0
             88 1962161   15     0
             89 1605442   15     0
             90 2166096   15     0
             91 1777296   15     0
             92 1232281   15     0
             93 1328833   15     0
             94 1288621   15     0
             95 1129704   15     0
 [ eached 'max' / getOption("max.print") -- omitted 98670 rows ]
> s b=data[c('Post.userid','Post.userage','likes')]
 E ror in `[.data.frame`(data, c("Post.userid", "Post.userage", "likes")) :undefined
 c lumns selected
> s b
 u erid age likes
            1  2094382   14     0
            2  1192601   14     0
            3  2083884   14     0
            4  1203168   14     0
            5  1733186   14     0
            6  1524765   14     0
            7  1136133   13     0
            8  1680361   13     0
            9  1365174   13     0
            10  1712567   13     0
            11  1612453   13     0
            12  2104073   13     0
            13  1918584   13     0
            14  1704433   13     0
            15  1932519   13     0
            16  1751722   13     0
            17  1470850   13     0
            18  1001768   13     0
            19  1537661   13     0
            20  1020296   13     0
```

```
21 1472643   13      0
22 2041297   13      0
23 1514978   13      0
24 1708962   15      0
25 1098955   15      0
26 1001243   15      0
27 2113084   15      0
28 2163454   15      0
29 1670750   15      0
30 1041376   15      0
31 1762274   15      0
32 1903650   14      0
33 2087235   14      0
34 1871735   14      0
35 1459785   14      0
36 1215208   14      0
37 1255528   14      0
38 1390333   14      0
39 1570689   14      0
40 1197377   14      0
41 1472999   14      0
42 1549614   14      0
43 2056802   14      0
44 2091823   14      0
45 1302193   14      0
46 1784528   14      0
47 1842851   14      0
48 1002019   14      0
49 2188166   14      0
50 1443541   14      0
51 1927282   14      0
52 1929885   14      0
53 2188851   14      0
54 1562606   14      0
55 1599485   14      0
56 2157419   14      0
57 1264260   14      0
58 1123633   14      0
59 2045259   14      0
60 1712642   14      0
61 1304645   14      0
62 1287843   14      0
63 1094094   14      0
64 1201176   14      0
65 1717785   14      0
66 1764393   14      0
67 1483573   14      0
68 2163813   16      0
```

```
69 1435172  16   0
70 1732320  16   0
71 1866176  16   0
72 1402504  16   0
73 1469691  16   0
74 1820036  15   0
75 1922550  15   0
76 1450870  15   0
77 2044883  15   0
78 2055501  15   0
79 1574525  15   0
80 1460963  15   0
81 1648391  15   0
82 2162779  15   0
83 1105499  15   0
84 1654852  15   0
85 1665635  15   0
86 2113685  15   0
87 1772186  15   0
88 1962161  15   0
89 1605442  15   0
90 2166096  15   0
91 1777296  15   0
92 1232281  15   0
93 1328833  15   0
94 1288621  15   0
95 1129704  15   0
96 1126847  15   0
97 1056469  15   0
98 2190921  15   0
99 1496299  15   0
100 1297545 15     0
101 1314465 15     0
102 1832374 15     0
103 2164634 15     0
104 1012962 15     0
105 1294086 15     0
106 1160530 15     0
107 1859444 15     0
108 1757068 15     0
109 1998492 15     0
110 1536686 15     0
111 2070224 15     0
112 1608131 15     0
113 1483131 17     0
114 1677748 17     0
115 1703844 17     0
116 1695626 17     0
```

```
      117 1054620 17    0
      118 1885663 16    0
      119 1174983 16    0
      120 1300831 16    0
      121 1106680 16    0
      122 1712570 16    0
      123 1505398 16    0
      124 2071007 16    0
      125 1387870 16    0
      126 1997182 16    0
      127 1409046 16    0
      128 1020864 16    0
      129 1662734 16    0
      130 1917166 16    0
      131 1689737 16    0
      132 1011372 16    0
      133 2191252 16    0
      134 2064115 16    0
      135 1449467 16    0
      136 1505097 16    0
      137 1306187 16    0
      138 1430230 16    0
      139 2126242 16    0
      140 1096123 16    0
      141 1733413 16    0
      142 2153790 16    0
      143 1639871 16    0
      144 1542192 16    0
      145 1664129 16    0
[ reached 'max' / getOption("max.print") -- omitted 98670 rows ]
> hea
d(s
ub)
user
id
age
like
s
      1 2094382 14    0
      2 1192601 14    0
      3 2083884 14    0
      4 1203168 14    0
      5 1733186 14    0
      6 1524765 14    0
```

CONCLUSION:

We have performing data preprocessing operation using R language

ASSIGNMENT NUMBER 2

TITLE: Create user login form using TKinter python library with mysqldatabaseconnectivity

Learning Objective:

GUI frame wrok designing

THEORY:

Tkinter is one of the Python libraries which contains many functions for the development of graphic user interface pages and windows. Login pages are importantfor the development of any kind of mobile or web application. This page is most essential for user authentication purposes.

We will use the mysql.connector library to establish a connection between Python projectand MySQL workbench. Db is the object created using mysql.connector.connect class which stores all the information about databases such database name, password, and table name.

Exercise:

```
import tkinter as tk
import mysql.connector from tkinter
 import * import PIL
def submitact():

user = Username.get() passw = password.get()

print(f"The name entered by you is {user} {passw}") logintodb(user, passw)

def logintodb(user, passw):

# If paswword is enetered by the # user
if passw:
db = mysql.connector.connect(host ="localhost", user = user,
password = passw, db ="College")
cursor = db.cursor()

# If no password is enetered by the # user
else:
db = mysql.connector.connect(host ="localhost", user = user,
db ="College") cursor = db.cursor()

# A Table in the database
```

```
savequery = "select * from STUDENT"

try:
cursor.execute(savequery) myresult = cursor.fetchall()
# Printing the result of the # query
for x in myresult: print(x)
print("Query Excecuted successfully")

except:
db.rollback() print("Error occured")


root = tk.Tk() root.geometry("300x300") root.title("DBMS Login Page")
C = Canvas(root, bg ="blue", height = 250, width = 300) # Definging the first row
lblfrstrow = tk.Label(root, text ="Username -", ) lblfrstrow.place(x = 50, y = 20)

Username = tk.Entry(root, width = 35) Username.place(x = 150, y = 20, width = 100)

lblsecrow = tk.Label(root, text ="Password -") lblsecrow.place(x = 50, y = 50)

password = tk.Entry(root, width = 35) password.place(x = 150, y = 50, width = 100)

submitbtn = tk.Button(root, text ="Login",
bg ='blue', command = submitact) submitbtn.place(x = 150, y = 135, width = 55)

root.mainloop()
```

CONCLUSION:

We have created user login window with mysql database connectivity.