PRACTICAL JOURNAL

in

PROGRAMMING WITH HADOOP AND SPARK

Submitted by

KSMSCIT010 HIMANSHU SINGH

for the award of the Degree of

MASTERS OF SCIENCE (INFORMATION TECHNOLOGY)

PART – II

DEPARTMENT OF INFORMATION TECHNOLOGY
KISHINCHAND CHELLARAM COLLEGE
(Affiliated to University of HSNCU)
MUMBAI,400020
MAHARASHTRA
2024-25

SUBJECT CODE – BIT616D

Programming with Hadoop and Spark

KISHINCHAND CHELLARAM COLLEGE

CHURCHGATE, MUMBAI – 400 020.

DEPARTMENT OF INFORMATION TECHNOLOGY

M.SC.I. T PART- II

CERTIFICATE

This is to certify that the Practical conducted by Mr. **HIMANSHU SINGH** for M.Sc. (IT) Part- II Semester- IV, Seat No: **KSMSCIT010** at Kishinchand Chellaram College in partial fulfillment for the MASTERS OF SCIENCE (INFORMATION TECHNOLOGY). Degree Examination for semester IV has been periodically examined and signed, and the course of term work has been satisfactorily carried out for the year 2024 - 2025. This Practical journal had not been submitted for any other examination and does not form part of any other course undergone by the candidate.

Signature	**Signature**	**Signature**
Lecturer-In-Charge	**External Examiner**	**Course Coordination**
Guided By	**Examined By**	**Certified By**

College Stamp

KISHINCHAND CHELLARAM COLLEGE, MUMBAI - 20

M.Sc. (I.T.) Part-2 Semester IV

INDEX

Sr. No	Topic	Date	Sign
1	A. Installation of Hadoop	29|12|2024	
	B. Basic Commands of Hadoop	12|01|2025	
2	PySpark Data Reading and Display Example	19|01|2025	
3	Combining Data Frames with PySpark	02/02/2025	
4	Collect, filter, map, MapReduce example using PySpark	09|02|2025	
5	Creating a spark session using the configuration and Data frame Creation	23|02|2025	
6	Creating a temporary view of the data frame to use SQL query with Spark Session	02/02/2025	
7	Creating User Defined Function with spark session	02|03|2025	
8	PySpark using MLlib library working with Linear Regression	23|03|2025	
9	PySpark using MLlib library working with KMeans Clustering	29|03|2025	
10	PySpark using MLlib library working with Logistic Regression	29|03|2025	
11	PySpark using MLlib library working with Naive Bayes	29|03|2025	
12	Real-Time Word Count with PySpark Streaming	29|03|2025	
13	PySpark Word Count and Data Manipulation Example	29|03|2025	
14	SLEP (Predictive Analysis of Security Threats in Independent India: A Machine Learning Approach to Identifying High-Risk States Using K-Means Clustering)	09|03|2025	
15	ASSIGNMENT	29|03|2025	
16	CASE STUDY	29|03|2025	

KISHINCHAND CHELLARAM COLLEGE, MUMBAI - 20

M.Sc. (I.T.) Part-2 Semester IV

Practical 1 - A

Hadoop Installation

Aim: Hadoop Installation.

Requirements:

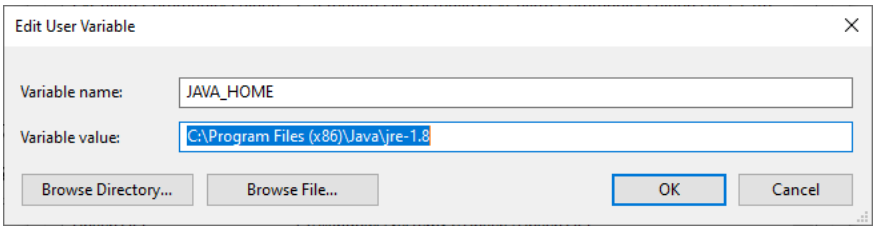
1. Java SE Runtime Environment Downloads: https://www.oracle.com/technetwork/ja... / https://drive.google.com/file/d/1kc2hL9Q4huX179FWJP3C84bkWXiBrGPE/view
2. Hadoop: https://hadoop.apache.org/releases.html

- **Windows Installation Steps:**

Step-1: Download and install Java 8, then verify the installation by typing java -version in the command prompt.

```
C:\Windows\system32>java -version
java version "1.8.0_202"
Java(TM) SE Runtime Environment (build 1.8.0_202-b08)
Java HotSpot(TM) 64-Bit Server VM (build 25.202-b08, mixed mode)
```

Step-2: Add Java to the environment variables by editing the user variables. Include the path where the Java files are saved.



To verify if Java is set as an environment variable, type `echo %JAVA_HOME%` in the command prompt.

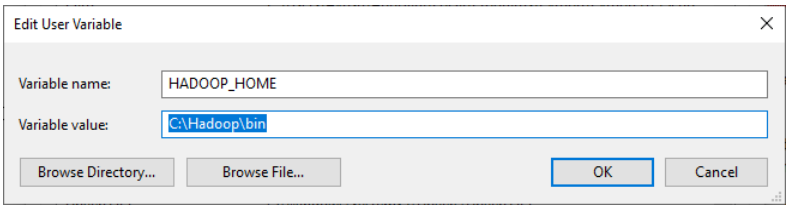
```
C:\Windows\system32>echo %JAVA_HOME%
C:\Program Files (x86)\Java\jre-1.8
```

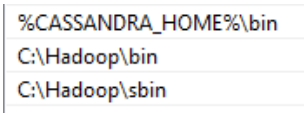
Step-3: Download and extract the Hadoop files. Navigate to `etc -> hadoop`, find `hadoop-env.cmd`, right-click, and edit the file. Update the Java path **(Note: use `Progra~1` instead of "Program Files" to avoid issues)**. Set this path in the environment variables and verify the setup by typing `hadoop version` in the command prompt.

- ***(Setting up path in hadoop-env.cmd)***

```
@rem The java implementation to use.  Required.
set JAVA_HOME=C:\Progra~1\Java\jdk1.8.0_202
```

- ***(Setting up path in environment variables)***





- ***(Checking Hadoop)***

```
C:\Users\Ayush>hadoop version
Hadoop 3.4.0
Source code repository git@github.com:apache/hadoop.git -r bd8b77f398f626bb7791783192ee7a5dfaeec760
Compiled by root on 2024-03-04T06:35Z
Compiled on platform linux-x86_64
Compiled with protoc 3.21.12
From source with checksum f7fe694a3613358b38812ae9c31114e
This command was run using /C:/Hadoop/share/hadoop/common/hadoop-common-3.4.0.jar
```

Step-4: Go to these files and edit the properties.

- ***(file-> etc-> hadoop-> core-edit.xml file)***

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

- ***(Create a new folder named `data` inside the Hadoop folder. Inside `data`, create two folders named `datanode` and `namenode`. Then, open the `httpfs-site.xml` file located in `etc -> hadoop`, edit the properties, and add the paths for `namenode` and `datanode`.)***

```
<configuration>
<property>
  <name>dfs.replication</name>
  <value>1</value>
</property><property>
  <name>dfs.namenode.name.dir</name>
  <value>C:\Hadoop\data\namenode</value>
</property><property>
  <name>dfs.datanode.data.dir</name>
  <value>C:\Hadoop\data\datanode</value>
</property>
</configuration>
```

- ***(file-> etc-> hadoop-> mapred-site.xml)***

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

- ***(file-> etc-> hadoop-> 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>
```

Step-5: Delete the existing `bin` folder and replace it with the fixed `bin` folder.

Link: https://drive.google.com/file/d/1nCN_jK7EJF2DmPUUxgOggnvJ6k6tksYz/view

Step-6: Open Command Prompt as Administrator and run the command `hdfs namenode -format`.

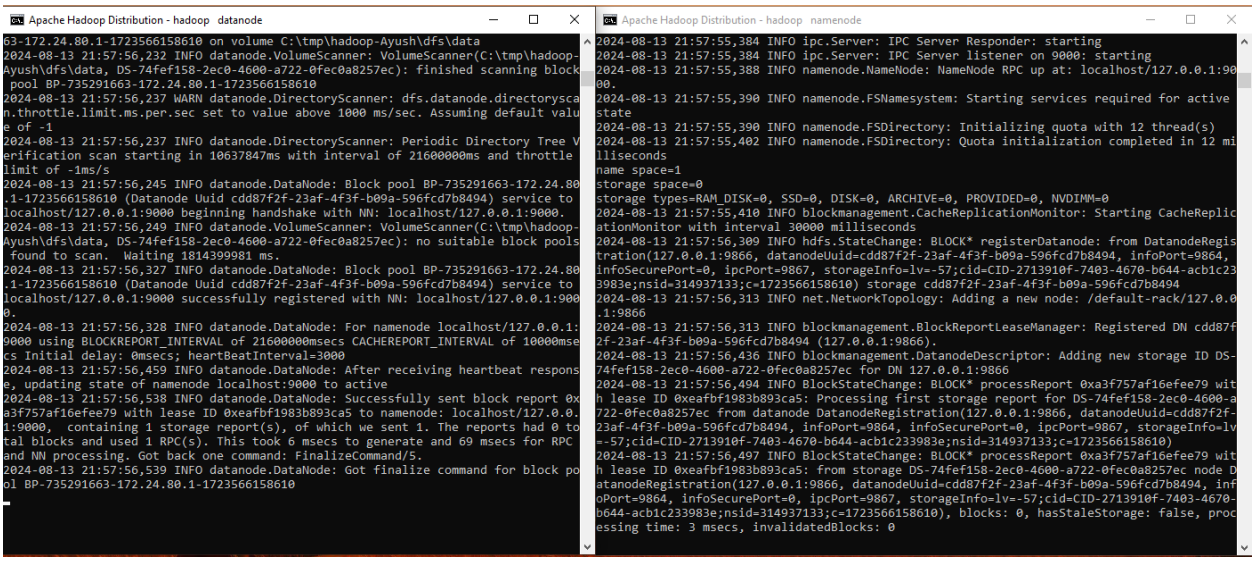
```
C:\Windows\system32>hdfs namenode -format
```

It will start the NameNode

Change the path and type start-dfs.cmd

```
C:\Windows\system32>cd \

C:\>cd Hadoop

C:\Hadoop>cd sbin

C:\Hadoop\sbin>start-dfs.cmd
```

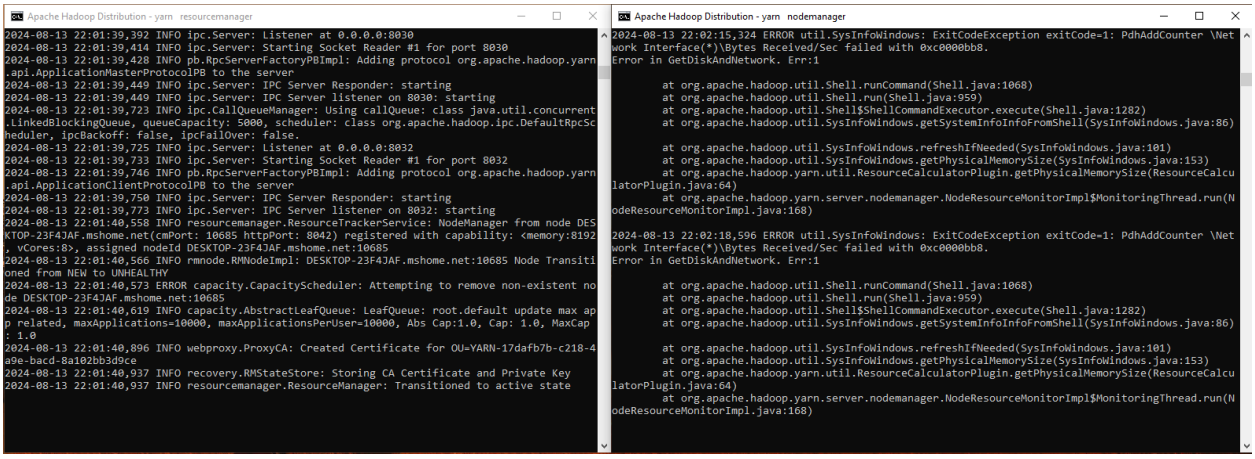
After pressing Enter, Hadoop will start.



To check the running status type “jps”

```
C:\Hadoop\sbin>jps
16160 Jps
5824 NameNode
15916 DataNode
```

Now, type start-yarn.cmd. This will start the ResourceManager.



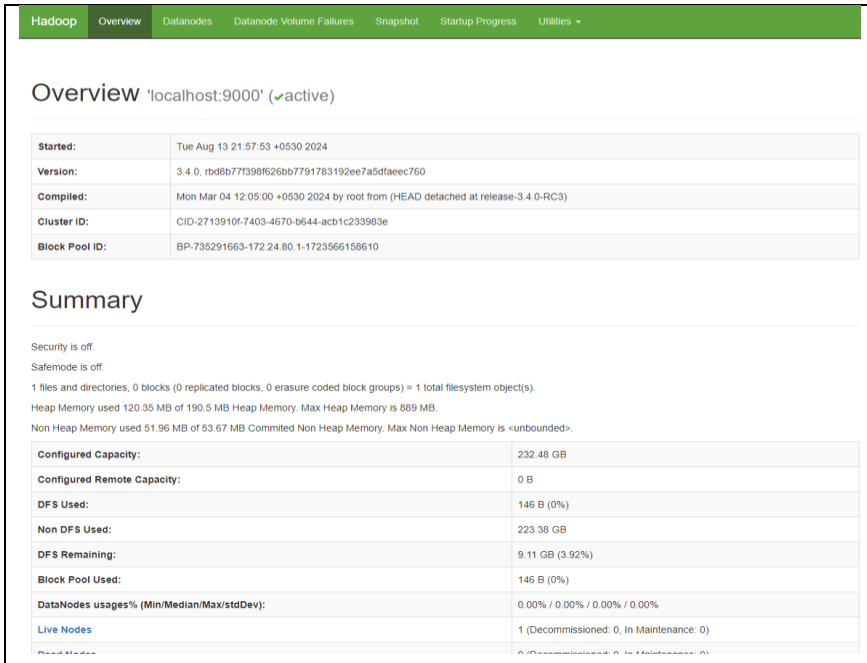
```
Apache Hadoop Distribution - yarn resourcemanager
2024-08-13 22:01:39,392 INFO ipc.Server: Listener at 0.0.0.0:8030
2024-08-13 22:01:39,414 INFO ipc.Server: Starting Socket Reader #1 for port 8030
2024-08-13 22:01:39,428 INFO pb.RpcServerFactoryPBImpl: Adding protocol org.apache.hadoop.yarn
.api.ApplicationMasterProtocolPB to the server
2024-08-13 22:01:39,449 INFO ipc.Server: IPC Server Responder: starting
2024-08-13 22:01:39,449 INFO ipc.Server: IPC Server listener on 8030: starting
2024-08-13 22:01:39,723 INFO ipc.CallQueueManager: Using callQueue: class java.util.concurrent
.LinkedBlockingQueue, queueCapacity: 5000, scheduler: class org.apache.hadoop.ipc.DefaultRpcSc
heduler, ipcBackoff: false, ipcFailOver: false.
2024-08-13 22:01:39,725 INFO ipc.Server: Listener at 0.0.0.0:8032
2024-08-13 22:01:39,733 INFO ipc.Server: Starting Socket Reader #1 for port 8032
2024-08-13 22:01:39,746 INFO pb.RpcServerFactoryPBImpl: Adding protocol org.apache.hadoop.yarn
.api.ApplicationClientProtocolPB to the server
2024-08-13 22:01:39,750 INFO ipc.Server: IPC Server Responder: starting
2024-08-13 22:01:39,773 INFO ipc.Server: IPC Server listener on 8032: starting
2024-08-13 22:01:40,558 INFO resourcemanager.ResourceTrackerService: NodeManager from node DES
KTOP-23F4JAF.mshome.net(cmPort: 10685 httpPort: 8042) registered with capability: <memory:8192
, vCores:8>, assigned nodeId DESKTOP-23F4JAF.mshome.net:10685
2024-08-13 22:01:40,566 INFO rmnode.RMNodeImpl: DESKTOP-23F4JAF.mshome.net:10685 Node Transiti
oned from NEW to UNHEALTHY
2024-08-13 22:01:40,573 ERROR capacity.CapacityScheduler: Attempting to remove non-existent no
de DESKTOP-23F4JAF.mshome.net:10685
2024-08-13 22:01:40,619 INFO capacity.AbstractLeafQueue: LeafQueue: root.default update max ap
p related, maxApplications=10000, maxApplicationsPerUser=10000, Abs Cap:1.0, Cap: 1.0, MaxCap
: 1.0
2024-08-13 22:01:40,896 INFO webproxy.ProxyCA: Created Certificate for OU=YARN-17dafb7b-c218-4
a9e-bacd-8a102bb3d9ce
2024-08-13 22:01:40,937 INFO recovery.RMStateStore: Storing CA Certificate and Private Key
2024-08-13 22:01:40,937 INFO resourcemanager.ResourceManager: Transitioned to active state
```

```
Apache Hadoop Distribution - yarn nodemanager
2024-08-13 22:02:15,324 ERROR util.SysInfoWindows: ExitCodeException exitCode=1: PdhAddCounter \Net
work Interface(*)\Bytes Received/Sec failed with 0xc0000bb8.
Error in GetDiskAndNetwork. Err:1

        at org.apache.hadoop.util.Shell.runCommand(Shell.java:1068)
        at org.apache.hadoop.util.Shell.run(Shell.java:959)
        at org.apache.hadoop.util.Shell$ShellCommandExecutor.execute(Shell.java:1282)
        at org.apache.hadoop.util.SysInfoWindows.getSystemInfoInfoFromShell(SysInfoWindows.java:86)

        at org.apache.hadoop.util.SysInfoWindows.refreshIfNeeded(SysInfoWindows.java:101)
        at org.apache.hadoop.util.SysInfoWindows.getPhysicalMemorySize(SysInfoWindows.java:153)
        at org.apache.hadoop.yarn.util.ResourceCalculatorPlugin.getPhysicalMemorySize(ResourceCalcu
latorPlugin.java:64)
        at org.apache.hadoop.yarn.server.nodemanager.NodeResourceMonitorImpl$MonitoringThread.run(N
odeResourceMonitorImpl.java:168)

2024-08-13 22:02:18,596 ERROR util.SysInfoWindows: ExitCodeException exitCode=1: PdhAddCounter \Net
work Interface(*)\Bytes Received/Sec failed with 0xc0000bb8.
Error in GetDiskAndNetwork. Err:1

        at org.apache.hadoop.util.Shell.runCommand(Shell.java:1068)
        at org.apache.hadoop.util.Shell.run(Shell.java:959)
        at org.apache.hadoop.util.Shell$ShellCommandExecutor.execute(Shell.java:1282)
        at org.apache.hadoop.util.SysInfoWindows.getSystemInfoInfoFromShell(SysInfoWindows.java:86)

        at org.apache.hadoop.util.SysInfoWindows.refreshIfNeeded(SysInfoWindows.java:101)
        at org.apache.hadoop.util.SysInfoWindows.getPhysicalMemorySize(SysInfoWindows.java:153)
        at org.apache.hadoop.yarn.util.ResourceCalculatorPlugin.getPhysicalMemorySize(ResourceCalcu
latorPlugin.java:64)
        at org.apache.hadoop.yarn.server.nodemanager.NodeResourceMonitorImpl$MonitoringThread.run(N
odeResourceMonitorImpl.java:168)
```

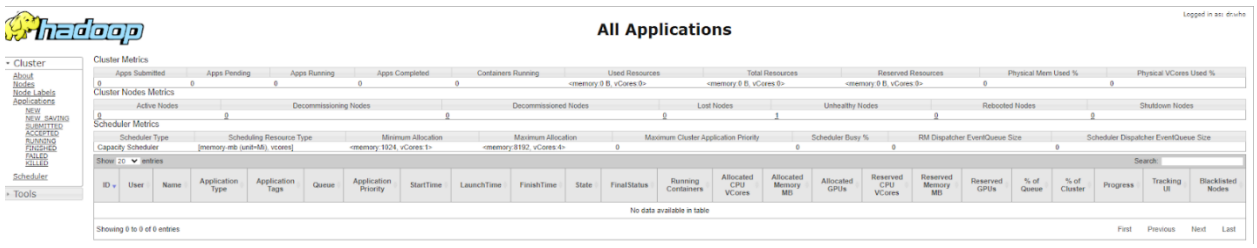
To check the running status type “jps”

```
C:\Hadoop\sbin>jps
15824 ResourceManager
5824 NameNode
8048 NodeManager
11980 Jps
15916 DataNode
```

Step-6: To check the running status of Hadoop, open Chrome and type `localhost:9870`.



To check the **Results Manager** of Hadoop, open Chrome and type **'localhost:8088'.**



Step-7: To stop the running status and processes, type stop-all.cmd

```
C:\Hadoop\sbin>stop-all.cmd
This script is Deprecated. Instead use stop-dfs.cmd and stop-yarn.cmd
SUCCESS: Sent termination signal to the process with PID 15536.
SUCCESS: Sent termination signal to the process with PID 15956.
stopping yarn daemons
SUCCESS: Sent termination signal to the process with PID 12984.
SUCCESS: Sent termination signal to the process with PID 16120.

INFO: No tasks running with the specified criteria.
```

Linux Installation Steps:

Step-1: First, update the package list. Then, install the OpenJDK package and type java -version to verify.

```
┌──(kali㉿kali)-[~]
└─$ java -version
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
openjdk version "21.0.3" 2024-04-16
OpenJDK Runtime Environment (build 21.0.3+9-Debian-2)
OpenJDK 64-Bit Server VM (build 21.0.3+9-Debian-2, mixed mode, sharing)

┌──(kali㉿kali)-[~]
└─$ which javac
javac not found

┌──(kali㉿kali)-[~]
└─$ sudo apt update
[sudo] password for kali:
Get:1 http://kali.download/kali kali-rolling InRelease [41.5 kB]
Get:2 http://kali.download/kali kali-rolling/main amd64 Packages [19.9 MB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [47.8 MB]
Get:4 http://kali.download/kali kali-rolling/contrib amd64 Packages [109 kB]
Get:5 http://kali.download/kali kali-rolling/contrib amd64 Contents (deb) [267 kB]
Get:6 http://kali.download/kali kali-rolling/non-free amd64 Packages [194 kB]
Get:7 http://kali.download/kali kali-rolling/non-free amd64 Contents (deb) [863 kB]
Get:8 http://kali.download/kali kali-rolling/non-free-firmware amd64 Packages [33.1 kB]
Get:9 http://kali.download/kali kali-rolling/non-free-firmware amd64 Contents (deb) [16.9 kB]
Fetched 69.2 MB in 19s (3,715 kB/s)
847 packages can be upgraded. Run 'apt list --upgradable' to see them.

┌──(kali㉿kali)-[~]
└─$ sudo apt install default-jdk
Upgrading:
  openjdk-17-jre openjdk-17-jre-headless

Installing:
  default-jdk

Installing dependencies:
  default-jdk-headless openjdk-17-jdk openjdk-17-jdk-headless

Suggested packages:
  openjdk-17-demo openjdk-17-source visualvm

Summary:
  Upgrading: 2, Installing: 4, Removing: 0, Not Upgrading: 845
  Download size: 118 MB
  Space needed: 81.6 MB / 64.7 GB available

Continue? [Y/n] y
```

```
┌──(kali㉿kali)-[~]
└─$ java -version
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
openjdk version "21.0.3" 2024-04-16
OpenJDK Runtime Environment (build 21.0.3+9-Debian-2)
OpenJDK 64-Bit Server VM (build 21.0.3+9-Debian-2, mixed mode, sharing)

┌──(kali㉿kali)-[~]
└─$ which javac
/usr/bin/javac

┌──(kali㉿kali)-[~]
└─$ sudo adduser hadoop
info: Adding user `hadoop' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `hadoop' (1001) ...
info: Adding new user `hadoop' (1001) with group `hadoop (1001)' ...
info: Creating home directory `/home/hadoop' ...
info: Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for hadoop
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
info: Adding new user `hadoop' to supplemental / extra groups `users' ...
info: Adding user `hadoop' to group `users' ...
```

Step-2: Create a dedicated user for Hadoop to run services and manage administrative tasks.

To add a new user called hadoop, we use the adduser command

```
┌──(kali㉿kali)-[~]
└─$ java -version
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
openjdk version "21.0.3" 2024-04-16
OpenJDK Runtime Environment (build 21.0.3+9-Debian-2)
OpenJDK 64-Bit Server VM (build 21.0.3+9-Debian-2, mixed mode, sharing)

┌──(kali㉿kali)-[~]
└─$ which javac
/usr/bin/javac

┌──(kali㉿kali)-[~]
└─$ sudo adduser hadoop
info: Adding user `hadoop' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `hadoop' (1001) ...
info: Adding new user `hadoop' (1001) with group `hadoop (1001)' ...
info: Creating home directory `/home/hadoop' ...
info: Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for hadoop
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
info: Adding new user `hadoop' to supplemental / extra groups `users' ...
info: Adding user `hadoop' to group `users' ...
```

Add the Hadoop user to the sudo group to grant administrative privileges.

```
┌──(kali㉿kali)-[~]
└─$ sudo usermod -aG sudo hadoop

┌──(kali㉿kali)-[~]
└─$ sudo su - hadoop
┌──(hadoop㉿kali)-[~]
└─$ sudo apt install openssh-server openssh-client -y
[sudo] password for hadoop:
Upgrading:
  openssh-client openssh-server openssh-sftp-server

Summary:
  Upgrading: 3, Installing: 0, Removing: 0, Not Upgrading: 842
  Download size: 1,488 kB
  Space needed: 1,024 B / 64.6 GB available

Get:2 http://kali.download/kali kali-rolling/main amd64 openssh-server amd64 1:9.7p1-7 [458 kB]
Get:1 http://mirror.kku.ac.th/kali kali-rolling/main amd64 openssh-sftp-server amd64 1:9.7p1-7 [65.1 kB]
Get:3 http://mirror.freedif.org/kali kali-rolling/main amd64 openssh-client amd64 1:9.7p1-7 [964 kB]
Fetched 1,488 kB in 2s (769 kB/s)
Preconfiguring packages ...
(Reading database ... 391256 files and directories currently installed.)
Preparing to unpack .../openssh-sftp-server_1%3a9.7p1-7_amd64.deb ...
Unpacking openssh-sftp-server (1:9.7p1-7) over (1:9.7p1-5) ...
Preparing to unpack .../openssh-server_1%3a9.7p1-7_amd64.deb ...
Unpacking openssh-server (1:9.7p1-7) over (1:9.7p1-5) ...
Preparing to unpack .../openssh-client_1%3a9.7p1-7_amd64.deb ...
Unpacking openssh-client (1:9.7p1-7) over (1:9.7p1-5) ...
Setting up openssh-client (1:9.7p1-7) ...
Setting up openssh-sftp-server (1:9.7p1-7) ...
Setting up openssh-server (1:9.7p1-7) ...
Installing new version of config file /etc/pam.d/sshd ...
ssh.service is a disabled or a static unit not running, not starting it.
ssh.socket is a disabled or a static unit not running, not starting it.
Processing triggers for kali-menu (2023.4.7) ...
Processing triggers for man-db (2.12.1-1) ...
```

Switch to the newly created Hadoop user to ensure all subsequent commands run with the appropriate permissions.

```
┌──(kali㉿kali)-[~]
└─$ sudo usermod -aG sudo hadoop

┌──(kali㉿kali)-[~]
└─$ sudo su - hadoop
┌──(hadoop㉿kali)-[~]
└─$ sudo apt install openssh-server openssh-client -y
[sudo] password for hadoop:
Upgrading:
  openssh-client openssh-server openssh-sftp-server

Summary:
  Upgrading: 3, Installing: 0, Removing: 0, Not Upgrading: 842
  Download size: 1,488 kB
  Space needed: 1,024 B / 64.6 GB available

Get:2 http://kali.download/kali kali-rolling/main amd64 openssh-server amd64 1:9.7p1-7 [458 kB]
Get:1 http://mirror.kku.ac.th/kali kali-rolling/main amd64 openssh-sftp-server amd64 1:9.7p1-7 [65.1 kB]
Get:3 http://mirror.freedif.org/kali kali-rolling/main amd64 openssh-client amd64 1:9.7p1-7 [964 kB]
Fetched 1,488 kB in 2s (769 kB/s)
Preconfiguring packages ...
(Reading database ... 391256 files and directories currently installed.)
Preparing to unpack .../openssh-sftp-server_1%3a9.7p1-7_amd64.deb ...
Unpacking openssh-sftp-server (1:9.7p1-7) over (1:9.7p1-5) ...
Preparing to unpack .../openssh-server_1%3a9.7p1-7_amd64.deb ...
Unpacking openssh-server (1:9.7p1-7) over (1:9.7p1-5) ...
Preparing to unpack .../openssh-client_1%3a9.7p1-7_amd64.deb ...
Unpacking openssh-client (1:9.7p1-7) over (1:9.7p1-5) ...
Setting up openssh-client (1:9.7p1-7) ...
Setting up openssh-sftp-server (1:9.7p1-7) ...
Setting up openssh-server (1:9.7p1-7) ...
Installing new version of config file /etc/pam.d/sshd ...
ssh.service is a disabled or a static unit not running, not starting it.
ssh.socket is a disabled or a static unit not running, not starting it.
Processing triggers for kali-menu (2023.4.7) ...
Processing triggers for man-db (2.12.1-1) ...
```

Step-3: Configure password-less SSH access to enable seamless communication between nodes in the Hadoop cluster, allowing Hadoop daemons to securely communicate without manual authentication.

First, we install the OpenSSH server and client

```
┌──(kali㉿kali)-[~]
└─$ sudo usermod -aG sudo hadoop

┌──(kali㉿kali)-[~]
└─$ sudo su - hadoop
┌──(hadoop㉿kali)-[~]
└─$ sudo apt install openssh-server openssh-client -y
[sudo] password for hadoop:
Upgrading:
  openssh-client openssh-server openssh-sftp-server

Summary:
  Upgrading: 3, Installing: 0, Removing: 0, Not Upgrading: 842
  Download size: 1,488 kB
  Space needed: 1,024 B / 64.6 GB available

Get:2 http://kali.download/kali kali-rolling/main amd64 openssh-server amd64 1:9.7p1-7 [458 kB]
Get:1 http://mirror.kku.ac.th/kali kali-rolling/main amd64 openssh-sftp-server amd64 1:9.7p1-7 [65.1 kB]
Get:3 http://mirror.freedif.org/kali kali-rolling/main amd64 openssh-client amd64 1:9.7p1-7 [964 kB]
Fetched 1,488 kB in 2s (769 kB/s)
Preconfiguring packages ...
(Reading database ... 391256 files and directories currently installed.)
Preparing to unpack .../openssh-sftp-server_1%3a9.7p1-7_amd64.deb ...
Unpacking openssh-sftp-server (1:9.7p1-7) over (1:9.7p1-5) ...
Preparing to unpack .../openssh-server_1%3a9.7p1-7_amd64.deb ...
Unpacking openssh-server (1:9.7p1-7) over (1:9.7p1-5) ...
Preparing to unpack .../openssh-client_1%3a9.7p1-7_amd64.deb ...
Unpacking openssh-client (1:9.7p1-7) over (1:9.7p1-5) ...
Setting up openssh-client (1:9.7p1-7) ...
Setting up openssh-sftp-server (1:9.7p1-7) ...
Setting up openssh-server (1:9.7p1-7) ...
Installing new version of config file /etc/pam.d/sshd ...
ssh.service is a disabled or a static unit not running, not starting it.
ssh.socket is a disabled or a static unit not running, not starting it.
Processing triggers for kali-menu (2023.4.7) ...
Processing triggers for man-db (2.12.1-1) ...
```

This command installs OpenSSH on the Linux system and ensures the server and client components are available.Additionally, by default, the OpenSSH service is started automatically after installation. Let's verify the SSH service status

```
┌──(hadoop㉿kali)-[~]
└─$ systemctl start ssh

┌──(hadoop㉿kali)-[~]
└─$ systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/usr/lib/systemd/system/ssh.service; disabled; preset: disabled)
     Active: active (running) since Wed 2024-08-21 13:12:41 EDT; 2s ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 6270 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 6274 (sshd)
      Tasks: 1 (limit: 2272)
     Memory: 1.5M (peak: 1.8M)
        CPU: 39ms
     CGroup: /system.slice/ssh.service
             └─6274 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
```

Step-4: Generate SSH Key Pair.

Generate an RSA key pair with an empty passphrase, saving the private key in `~/.ssh/id_rsa` and the public key in `~/.ssh/id_rsa.pub`.

```
┌──(hadoop㉿kali)-[~]
└─$ ssh-keygen -t rsa -P "" -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
Created directory '/home/hadoop/.ssh'.
Your identification has been saved in /home/hadoop/.ssh/id_rsa
Your public key has been saved in /home/hadoop/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:57rNF25t+zO91ApV6TjJsSiJabYBP+Vkab8Q0I0e2/w hadoop@kali
The key's randomart image is:
+---[RSA 3072]----+
|        .. o     |
|         .+ ..   .|
|      .  .B=  . ..|
|      o Oo+oo *. |
|       OS=.o.*.. |
|      o +oo oE. .|
|       .  .o.o .o|
|         +  +.++.|
|        o.oo .o+=|
+----[SHA256]-----+
```

Copy the contents of the public key to the `authorized_keys` file on the local machine

```
┌──(kali㉿kali)-[~]
└─$ cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

Modify the permissions of the `~/.ssh` directory and the `authorized_keys` file to ensure proper access.

```
┌──(hadoop㉿kali)-[~]
└─$ chmod 700 ~/.ssh

┌──(hadoop㉿kali)-[~]
└─$ chmod 600 ~/.ssh/authorized_keys
```

Finally, to test the SSH connection, we leverage the ssh command

```
┌──(hadoop㉿kali)-[~]
└─$ ssh localhost
The authenticity of host 'localhost (::1)' can't be established.
ED25519 key fingerprint is SHA256:QgN2vAShEksvers6anv8xyTLlqvaDCx1T/yPsQ2dGAU.
This key is not known by any other names.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'localhost' (ED25519) to the list of known hosts.
Linux kali 6.6.15-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.6.15-2kali1 (2024-05-17) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
```

Step-5: Install Apache Hadoop

First, we log in as the hadoop user with the su command

```
┌──(hadoop㉿kali)-[~]
└─$ sudo su - hadoop
[sudo] password for hadoop:
┌──(hadoop㉿kali)-[~]
└─$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
--2024-08-21 13:15:14--  https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
Resolving downloads.apache.org (downloads.apache.org)... 88.99.208.237, 135.181.214.104, 2a01:4f8:10a:39da::2, ...
Connecting to downloads.apache.org (downloads.apache.org)|88.99.208.237|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 730107476 (696M) [application/x-gzip]
Saving to: 'hadoop-3.3.6.tar.gz'

hadoop-3.3.6.tar.gz                 100%[===================================================================================================>] 696.28M   425KB/s    in 12m 1

2024-08-21 13:27:33 (965 KB/s) - 'hadoop-3.3.6.tar.gz' saved [730107476/730107476]


┌──(hadoop㉿kali)-[~]
└─$ tar -xvzf hadoop-3.3.6.tar.gz
```

Download the latest stable version of Hadoop from the official Apache Hadoop download page using the `wget` command. Link: https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz


```
┌──(hadoop㉿kali)-[~]
└─$ sudo su - hadoop
[sudo] password for hadoop:
┌──(hadoop㉿kali)-[~]
└─$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
--2024-08-21 13:15:14--  https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
Resolving downloads.apache.org (downloads.apache.org)... 88.99.208.237, 135.181.214.104, 2a01:4f8:10a:39da::2, ...
Connecting to downloads.apache.org (downloads.apache.org)|88.99.208.237|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 730107476 (696M) [application/x-gzip]
Saving to: 'hadoop-3.3.6.tar.gz'

hadoop-3.3.6.tar.gz                 100%[===================================================================================================>] 696.28M   425KB/s    in 12m 1

2024-08-21 13:27:33 (965 KB/s) - 'hadoop-3.3.6.tar.gz' saved [730107476/730107476]


┌──(hadoop㉿kali)-[~]
└─$ tar -xvzf hadoop-3.3.6.tar.gz
```


Once the download is complete, we extract the downloaded file with the tar command

```
┌──(hadoop㉿kali)-[~]
└─$ sudo su - hadoop
[sudo] password for hadoop:
┌──(hadoop㉿kali)-[~]
└─$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
--2024-08-21 13:15:14--  https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
Resolving downloads.apache.org (downloads.apache.org)... 88.99.208.237, 135.181.214.104, 2a01:4f8:10a:39da::2, ...
Connecting to downloads.apache.org (downloads.apache.org)|88.99.208.237|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 730107476 (696M) [application/x-gzip]
Saving to: 'hadoop-3.3.6.tar.gz'

hadoop-3.3.6.tar.gz                 100%[===================================================================================>] 696.28M   425KB/s    in 12m 1

2024-08-21 13:27:33 (965 KB/s) - 'hadoop-3.3.6.tar.gz' saved [730107476/730107476]


┌──(hadoop㉿kali)-[~]
└─$ tar -xvzf hadoop-3.3.6.tar.gz
```

After downloading, extract the file using the `tar` command.

```
┌──(hadoop㉿kali)-[~]
└─$ sudo mv hadoop-3.3.6 /usr/local/hadoop
[sudo] password for hadoop:

┌──(hadoop㉿kali)-[~]
└─$ sudo mkdir /usr/local/hadoop/logs

┌──(hadoop㉿kali)-[~]
└─$ sudo chown -R hadoop:hadoop /usr/local/hadoop
```

Then, let's create a directory to store system logs for Hadoop

```
┌──(hadoop㉿kali)-[~]
└─$ sudo mv hadoop-3.3.6 /usr/local/hadoop
[sudo] password for hadoop:

┌──(hadoop㉿kali)-[~]
└─$ sudo mkdir /usr/local/hadoop/logs

┌──(hadoop㉿kali)-[~]
└─$ sudo chown -R hadoop:hadoop /usr/local/hadoop
```

Finally, we change the ownership of the Hadoop directory

```
┌──(hadoop㉿kali)-[~]
└─$ sudo mv hadoop-3.3.6 /usr/local/hadoop
[sudo] password for hadoop:

┌──(hadoop㉿kali)-[~]
└─$ sudo mkdir /usr/local/hadoop/logs

┌──(hadoop㉿kali)-[~]
└─$ sudo chown -R hadoop:hadoop /usr/local/hadoop
```

To configure the Hadoop environment variables, let's open the *~/.bashrc* file with a text editor

```
┌──(hadoop㉿kali)-[~]
└─$ sudo nano ~/.bashrc

┌──(hadoop㉿kali)-[~]
└─$ source ~/.bashrc
```

```
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_INSTALL=$HADOOP_HOME
export HADOOP_MAPRED_HOME=$HADOOP_HOME
export HADOOP_COMMON_HOME=$HADOOP_HOME
export HADOOP_HDFS_HOME=$HADOOP_HOME
export YARN_HOME=$HADOOP_HOME
export HADOOP_COMMON_LIB_NATIVE_DIR=$HADOOP_HOME/lib/native
export PATH=$PATH:$HADOOP_HOME/sbin:$HADOOP_HOME/bin
export HADOOP_OPTS="-Djava.library.path=$HADOOP_HOME/lib/native"
```

Finally, we activate the updated environment variables

```
┌──(hadoop㉿kali)-[~]
└─$ sudo nano ~/.bashrc

┌──(hadoop㉿kali)-[~]
└─$ source ~/.bashrc
```

Step-6: Configure Java Environment Variables

First, let's find the path to the Java compiler (*javac*):

```
┌──(kali㉿kali)-[~]
└─$ java -version
Picked up _JAVA_OPTIONS: -Dawt.useSystemAAFontSettings=on -Dswing.aatext=true
openjdk version "21.0.3" 2024-04-16
OpenJDK Runtime Environment (build 21.0.3+9-Debian-2)
OpenJDK 64-Bit Server VM (build 21.0.3+9-Debian-2, mixed mode, sharing)

┌──(kali㉿kali)-[~]
└─$ which javac
/usr/bin/javac

┌──(kali㉿kali)-[~]
└─$ sudo adduser hadoop
info: Adding user `hadoop' ...
info: Selecting UID/GID from range 1000 to 59999 ...
info: Adding new group `hadoop' (1001) ...
info: Adding new user `hadoop' (1001) with group `hadoop (1001)' ...
info: Creating home directory `/home/hadoop' ...
info: Copying files from `/etc/skel' ...
New password:
Retype new password:
passwd: password updated successfully
Changing the user information for hadoop
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
Is the information correct? [Y/n] Y
info: Adding new user `hadoop' to supplemental / extra groups `users' ...
info: Adding user `hadoop' to group `users' ...
```

Additionally, we use the *readlink* command to determine the directory OpenJDK is located

```
┌──(hadoop㉿kali)-[/usr/local/hadoop/lib]
└─$ readlink -f /usr/bin/javac
/usr/lib/jvm/java-17-openjdk-amd64/bin/javac
```

Next, we edit the *hadoop-env.sh* file to define the Java environment variables:

```
┌──(hadoop㉿kali)-[~]
└─$ sudo nano $HADOOP_HOME/etc/hadoop/hadoop-env.sh
```

```
export JAVA_HOME=/usr/lib/jvm/java-17-openjdk-amd64
export PATH=$JAVA_HOME/bin:$PATH
export HADOOP_CLASSPATH+=" $HADOOP_HOME/lib/*.jar"
```

Step-7: Configure Hadoop Environment

We start by navigating to the Hadoop *lib* directory:

```
┌──(kali㉿kali)-[~]
└─$ cd /usr/local/hadoop/lib
```

Then, we use the *wget* command to download the Javax activation file

```
┌──(kali㉿kali)-[/usr/local/hadoop/lib]
└─$ sudo wget https://jcenter.bintray.com/javax/activation/javax.activation-api/1.2.0/javax.activation-api-1.2.0.jar
[sudo] password for kali:
--2024-08-21 14:38:17--  https://jcenter.bintray.com/javax/activation/javax.activation-api/1.2.0/javax.activation-api-1.2.0.jar
Resolving jcenter.bintray.com (jcenter.bintray.com)... 18.232.172.199, 3.95.117.170, 18.214.194.113
Connecting to jcenter.bintray.com (jcenter.bintray.com)|18.232.172.199|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://repo1.maven.org/maven2/javax/activation/javax.activation-api/1.2.0/javax.activation-api-1.2.0.jar [following]
--2024-08-21 14:38:24--  https://repo1.maven.org/maven2/javax/activation/javax.activation-api/1.2.0/javax.activation-api-1.2.0.jar
Resolving repo1.maven.org (repo1.maven.org)... 199.232.192.209, 199.232.196.209, 2a04:4e42:4c::209, ...
Connecting to repo1.maven.org (repo1.maven.org)|199.232.192.209|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 56674 (55K) [application/java-archive]
Saving to: 'javax.activation-api-1.2.0.jar.1'

javax.activation-api-1.2.0.jar.1                    100%[===================================================]

2024-08-21 14:38:25 (204 KB/s) - 'javax.activation-api-1.2.0.jar.1' saved [56674/56674]
```

Once the download is complete, let's verify the installed Hadoop version

```
┌──(hadoop㉿kali)-[/usr/local/hadoop/lib]
└─$ hadoop version
Hadoop 3.3.6
Source code repository https://github.com/apache/hadoop.git -r 1be78238728da9266a4f88195058f08fd012bf9c
Compiled by ubuntu on 2023-06-18T08:22Z
Compiled on platform linux-x86_64
Compiled with protoc 3.7.1
From source with checksum 5652179ad55f76cb287d9c633bb53bbd
This command was run using /usr/local/hadoop/share/hadoop/common/hadoop-common-3.3.6.jar
```

Next, let's edit the *core-site.xml* configuration file to specify the URL for the NameNode

```
┌──(hadoop㉿kali)-[/usr/local/hadoop/lib]
└─$ sudo nano $HADOOP_HOME/etc/hadoop/core-site.xml
```

```
<configuration>
    <property>
        <name>fs.default.name</name>
        <value>hdfs://0.0.0.0:9000</value>
        <description>The default file system URI</description>
    </property>
</configuration>
```

Additionally, let's create a directory for storing node metadata and change the ownership to hadoop

```
┌──(hadoop㉿kali)-[/usr/local/hadoop/lib]
└─$ sudo mkdir -p /home/hadoop/hdfs/{namenode,datanode}

┌──(hadoop㉿kali)-[/usr/local/hadoop/lib]
└─$ sudo chown -R hadoop:hadoop /home/hadoop/hdfs
```

Edit the `hdfs-site.xml` configuration file to define the node metadata storage location and the replication factor.

```
┌──(hadoop㉿kali)-[/usr/local/hadoop/lib]
└─$ sudo nano $HADOOP_HOME/etc/hadoop/hdfs-site.xml
```

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

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

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

Additionally, we also edit the *mapred-site.xml* configuration file to define MapReduce values

```
┌──(hadoop㉿kali)-[/usr/local/hadoop/lib]
└─$ sudo nano $HADOOP_HOME/etc/hadoop/mapred-site.xml
```

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

Lastly, let's edit the *yarn-site.xml* configuration file and define YARN-related settings

```
┌──(hadoop㉿kali)-[/usr/local/hadoop/lib]
└─$ sudo nano $HADOOP_HOME/etc/hadoop/yarn-site.xml
```

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

While logged in as the Hadoop user, validate the Hadoop configuration and format the HDFS NameNode.

```
┌──(hadoop㉿kali)-[/usr/local/hadoop/lib]
└─$ hdfs namenode -format
2024-08-21 13:53:21,586 INFO namenode.NameNode: STARTUP_MSG:
/************************************************************
STARTUP_MSG: Starting NameNode
STARTUP_MSG:   host = kali/127.0.1.1
STARTUP_MSG:   args = [-format]
STARTUP_MSG:   version = 3.3.6
```

Step-8: Start the Apache Hadoop Cluster.

First, let's start the NameNode and DataNode via the *start-dfs.sh* script

```
┌──(hadoop㉿kali)-[/usr/local/hadoop/lib]
└─$ start-dfs.sh
Starting namenodes on [0.0.0.0]
0.0.0.0: Warning: Permanently added '0.0.0.0' (ED25519) to the list of known hosts.
Starting datanodes
Starting secondary namenodes [kali]
kali: Warning: Permanently added 'kali' (ED25519) to the list of known hosts.
```

Next, we start the YARN resource and node managers

```
┌──(hadoop㉿kali)-[/usr/local/hadoop/lib]
└─$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
```

After starting the Hadoop components, use the `jps` command to verify that they are running. This command lists all Java processes, including Hadoop components.

```
┌──(hadoop㉿kali)-[/usr/local/hadoop/lib]
└─$ jps
26834 DataNode
26712 NameNode
28650 Jps
27083 SecondaryNameNode
```

Step-9: Finally, we can open a Web browser to access Hadoop's NameNode (http://localhost:9870) and ResourceManager (http://localhost:8088) interfaces.

To check the running status of Hadoop, open Chrome and type `localhost:9870`.

Hadoop | Overview | Datanodes | Datanode Volume Failures | Snapshot | Startup Progress | Utilities

Overview '0.0.0.0:9000' (active)

Started:	Wed Aug 21 13:54:02 -0400 2024
Version:	3.3.6, r1be78238728da9266a4f88195058f08fd012bf9c
Compiled:	Sun Jun 18 04:22:00 -0400 2023 by ubuntu from (HEAD detached at release-3.3.6-RC1)
Cluster ID:	CID-0e48e052-8cbc-47c9-9d6b-c11aa2694557
Block Pool ID:	BP-1906468221-127.0.1.1-1724262805163

Summary

Security is off.

Safemode is off.

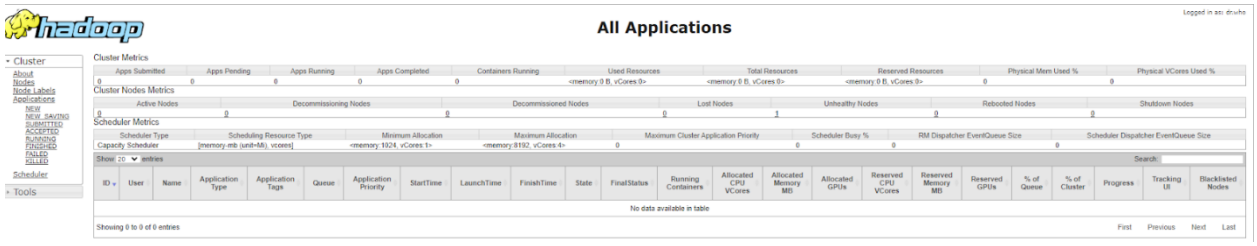
1 files and directories, 0 blocks (0 replicated blocks, 0 erasure coded block groups) = 1 total filesystem object(s).

Heap Memory used 34.33 MB of 60 MB Heap Memory. Max Heap Memory is 496 MB.

Non Heap Memory used 47.37 MB of 50 MB Commited Non Heap Memory. Max Non Heap Memory is <unbounded>.

Configured Capacity:	78.28 GB
Configured Remote Capacity:	0 B
DFS Used:	24 KB (0%)
Non DFS Used:	16.23 GB
DFS Remaining:	58.03 GB (74.13%)
Block Pool Used:	24 KB (0%)
DataNodes usages% (Min/Median/Max/stdDev):	0.00% / 0.00% / 0.00% / 0.00%
Live Nodes	1 (Decommissioned: 0, In Maintenance: 0)
Dead Nodes	0 (Decommissioned: 0, In Maintenance: 0)
Decommissioning Nodes	0
Entering Maintenance Nodes	0
Total Datanode Volume Failures	0 (0 B)
Number of Under-Replicated Blocks	0
Number of Blocks Pending Deletion (including replicas)	0
Block Deletion Start Time	Wed Aug 21 13:54:02 -0400 2024
Last Checkpoint Time	Wed Aug 21 13:53:25 -0400 2024
Enabled Erasure Coding Policies	RS-6-3-1024k

To check the Results Manager of Hadoop, open Chrome and type ‘localhost:8088’.



KISHINCHAND CHELLARAM COLLEGE, MUMBAI - 20

M.Sc. (I.T.) Part-2 Semester IV

Practical 1 - B

Basic Commands of Hadoop

Aim: Basic Commands of Hadoop

Basic commands of Hadoop

```
*************************************************************/
hadoop@248d42effe79:/usr/local/hadoop/lib$ start-dfs.sh
Starting namenodes on [0.0.0.0]
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known hosts.
Starting datanodes
Starting secondary namenodes [248d42effe79]
248d42effe79: Warning: Permanently added '248d42effe79,172.17.0.2' (ECDSA) to the list of known hosts.
hadoop@248d42effe79:/usr/local/hadoop/lib$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
hadoop@248d42effe79:/usr/local/hadoop/lib$ jps
17409 NameNode
18834 NodeManager
18020 SecondaryNameNode
19478 Jps
17625 DataNode
18619 ResourceManager
hadoop@248d42effe79:/usr/local/hadoop/lib$
```

1. Check HDFS Status

hdfs dfsadmin -report

This will display information about your Hadoop cluster, including storage details.

```
hadoop@248d42effe79:/usr/local/hadoop/lib$ hdfs dfsadmin -report
Configured Capacity: 105358065664 (98.12 GB)
Present Capacity: 75076349952 (69.92 GB)
DFS Remaining: 75076325376 (69.92 GB)
DFS Used: 24576 (24 KB)
DFS Used%: 0.00%
Replicated Blocks:
        Under replicated blocks: 0
        Blocks with corrupt replicas: 0
        Missing blocks: 0
        Missing blocks (with replication factor 1): 0
        Low redundancy blocks with highest priority to recover: 0
        Pending deletion blocks: 0
Erasure Coded Block Groups:
        Low redundancy block groups: 0
        Block groups with corrupt internal blocks: 0
        Missing block groups: 0
        Low redundancy blocks with highest priority to recover: 0
        Pending deletion blocks: 0

-------------------------------------------------
```

2. List Files in HDFS

hdfs dfs -ls /

This lists the files and directories in the root (/) of HDFS.

```
hadoop@248d42effe79:/usr/local/hadoop/lib$ hdfs dfs -ls /
Found 2 items
drwxr-xr-x   - hadoop supergroup          0 2025-04-01 14:08 /use
drwxr-xr-x   - hadoop supergroup          0 2025-04-01 14:08 /user
```

3. Create a Directory in HDFS

hdfs dfs -mkdir /user

hdfs dfs -mkdir /user/hadoop

This creates a directory named hadoop inside /user.

```
hadoop@248d42effe79:/usr/local/hadoop/lib$ hdfs dfs -mkdir /user/hadoop
hadoop@248d42effe79:/usr/local/hadoop/lib$ hdfs dfs -ls /
Found 2 items
drwxr-xr-x   - hadoop supergroup          0 2025-04-01 14:08 /use
drwxr-xr-x   - hadoop supergroup          0 2025-04-01 14:09 /user
hadoop@248d42effe79:/usr/local/hadoop/lib$
```

4. Upload a File to HDFS

If you have a file test.txt on your local machine, upload it to HDFS:

hdfs dfs -put C:\path\to\test.txt /user/hadoop/

Replace C:\path\to\test.txt with the actual file path.

```
vscode ➜ /workspaces/base-ubuntu (master) $ echo "Sohel Iqra Himanshu Shipra" > prac.txt
vscode ➜ /workspaces/base-ubuntu (master ✗) $ echo prac.txt
prac.txt
vscode ➜ /workspaces/base-ubuntu (master ✗) $ cat prac.txt
Sohel Iqra Himanshu Shipra
vscode ➜ /workspaces/base-ubuntu (master ✗) $ ^C
vscode ➜ /workspaces/base-ubuntu (master ✗) $
```

5. View File Contents in HDFS

hdfs dfs -cat /user/hadoop/test.txt

This displays the content of the file stored in HDFS.

```
hadoop@248d42effe79:/usr/local/hadoop/lib$ hdfs dfs -cat /user/hadoop/prac.txt
Sohel Iqra Himanshu Shipra
hadoop@248d42effe79:/usr/local/hadoop/lib$
```

KISHINCHAND CHELLARAM COLLEGE, MUMBAI - 20

M.Sc. (I.T.) Part-2 Semester IV

Practical 2

PySpark Data Reading and Display Example

Aim: PySpark Data Reading and Display Example.

Theory: PySpark, the Python API for Apache Spark, is widely used for big data processing. It provides a distributed computing framework that allows handling large datasets across multiple nodes. PySpark supports various data formats like CSV, JSON, and Parquet.

The read method in PySpark is used to load data from different sources, and the show() function is used to display a limited number of rows from the dataset. PySpark's lazy evaluation ensures efficient execution by optimizing the processing pipeline.

Objectives:

- To create a PySpark session for data processing.
- To read data from a CSV file into a PySpark DataFrame.
- To display the dataset and understand its structure.
- To select specific columns and analyze them.
- To convert the PySpark DataFrame into a Pandas DataFrame for further processing.

Code and Output:

```
!pip install pyspark
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.5)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7
```

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('read data eg').getOrCreate()
df = spark.read.option("Header",True).csv('Employee.csv')
df.show()
```

```
+------+--------+------+---+-------+
|emp_id|emp_name|salary|age|dept_id|
+------+--------+------+---+-------+
|     1|    Lisa| 25000| 22|    101|
|     2|   Jisso| 20000| 25|    102|
|     3|Taehyung| 40000| 27|    103|
|     4| Namjoon| 15000| 23|    104|
|     5|    Rose| 30000| 24|    105|
|     6|    Noah| 35000| 21|    106|
|     7|   Peter| 40000| 26|    107|
|     8|  Shipra| 50000| 22|    108|
|     9|    Moin| 45000| 21|    109|
|    10|     Sam| 10000| 23|    110|
+------+--------+------+---+-------+
```



```
df.show(5)
```

```
+------+--------+------+---+-------+
|emp_id|emp_name|salary|age|dept_id|
+------+--------+------+---+-------+
|     1|    Lisa| 25000| 22|    101|
|     2|   Jisso| 20000| 25|    102|
|     3|Taehyung| 40000| 27|    103|
|     4| Namjoon| 15000| 23|    104|
|     5|    Rose| 30000| 24|    105|
+------+--------+------+---+-------+
only showing top 5 rows
```

```
df.select("Salary").show()
```

```
+------+
|Salary|
+------+
| 25000|
| 20000|
| 40000|
| 15000|
| 30000|
| 35000|
| 40000|
| 50000|
| 45000|
| 10000|
+------+
```

```
df.select("Salary").show(3)
```

```
+------+
|Salary|
+------+
| 25000|
| 20000|
| 40000|
+------+
only showing top 3 rows
```

```
df.filter(df["Salary"]>15000).show()
```

```
+------+--------+------+---+-------+
|emp_id|emp_name|salary|age|dept_id|
+------+--------+------+---+-------+
|     1|    Lisa| 25000| 22|    101|
|     2|   Jisso| 20000| 25|    102|
|     3|Taehyung| 40000| 27|    103|
|     5|    Rose| 30000| 24|    105|
|     6|    Noah| 35000| 21|    106|
|     7|   Peter| 40000| 26|    107|
|     8|  Shipra| 50000| 22|    108|
|     9|    Moin| 45000| 21|    109|
+------+--------+------+---+-------+
```

****CONVERT THE PYSPARK DATA FRAME TO PANDAS****

```
pandas_df = df.toPandas()
print(pandas_df.head())
print(pandas_df.tail())
```

```
   emp_id  emp_name salary age dept_id
0       1      Lisa  25000  22     101
1       2     Jisso  20000  25     102
2       3  Taehyung  40000  27     103
3       4   Namjoon  15000  23     104
4       5      Rose  30000  24     105
   emp_id emp_name salary age dept_id
5       6     Noah  35000  21     106
6       7    Peter  40000  26     107
7       8   Shipra  50000  22     108
8       9     Moin  45000  21     109
9      10      Sam  10000  23     110
```

```
spark.stop()
```

Conclusion:

In this practical exercise, we successfully created a PySpark session and demonstrated how to read data from a CSV file into a DataFrame. We displayed the structure of the dataset and selected specific columns for analysis. Additionally, we converted the PySpark DataFrame into a Pandas DataFrame, allowing for further processing and analysis using the rich functionality of Pandas. This workflow highlights the versatility of PySpark in handling large datasets while leveraging the ease of use provided by Pandas for data manipulation.

KISHINCHAND CHELLARAM COLLEGE, MUMBAI - 20

M.Sc. (I.T.) Part-2 Semester IV

Practical 3 – A

Combining Data Frames with PySpark

Aim: Combining Data Frames with PySpark

Theory: In PySpark, combining or merging multiple DataFrames is a common operation performed using functions like:

- Union: Combines two DataFrames with the same schema (same column names and types).
- Join: Merges two DataFrames based on a common column (similar to SQL joins).
- Concatenation (with UnionByName): Merges DataFrames with different column orders but the same column names.
- Appending Rows/Columns: Adding new data to an existing DataFrame.

Merging DataFrames helps in data integration, feature enrichment, and combining datasets from different sources.

Objectives:

- Load multiple DataFrames from CSV files or other data sources.
- Combine DataFrames using different PySpark functions (union, join, etc.).
- Handle schema mismatches, missing values, and duplicates.
- Ensure efficient merging while maintaining data consistency.
- Analyze the final combined dataset.

Code and Output:

```
!pip install pyspark
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.5)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7
```

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
spark = SparkSession.builder.appName('Combined DataFrame').getOrCreate()
data1 = [(1,"Himanshu",50),(2,"Dhanraj",21),(3,"Sohel",5)]
data2 = [(4,"Shipra",22),(5,"Rashmi",25),(6,"Iqra",24)]
column = ["ID","Name","Age"]
df1 = spark.createDataFrame(data1,column)
df2 = spark.createDataFrame(data2,column)
df1.show()
df2.show()
```

```
+---+--------+---+
| ID|    Name|Age|
+---+--------+---+
|  1|Himanshu| 50|
|  2| Dhanraj| 21|
|  3|   Sohel|  5|
+---+--------+---+

+---+------+---+
| ID|  Name|Age|
+---+------+---+
|  4|Shipra| 22|
|  5|Rashmi| 25|
|  6|  Iqra| 24|
+---+------+---+
```

3 B - JOINS - Using UNION

```
# Using joins [merging dataframe on a column]

data4 = [(1,"Mumbai"),(2,"Bangalore"),(3,"Delhi"),(4,"Amritsar"),(5,"Kolkata"),(6,"Chennai")]

column2 = ["ID","City"]

df4 = spark.createDataFrame(data4,column2)
df4.show()
```

```
+---+---------+
| ID|     City|
+---+---------+
|  1|   Mumbai|
|  2|Bangalore|
|  3|    Delhi|
|  4| Amritsar|
|  5|  Kolkata|
|  6|  Chennai|
+---+---------+
```

KSMSCIT010- HIMANSHU SINGH

```
join_df = df1.join(df4,on="ID",how="inner")
join_df.show()
```

```
+---+--------+---+---------+
| ID|    Name|Age|     City|
+---+--------+---+---------+
|  1|Himanshu| 50|   Mumbai|
|  2| Dhanraj| 21|Bangalore|
|  3|   Sohel|  5|    Delhi|
+---+--------+---+---------+
```

Using JOIN (merging dataframe on column)

```
# Using joins [merging dataframe on a column]

data4 = [(1,"Mumbai"),(2,"Bangalore"),(3,"Delhi"),(4,"Amritsar"),(5,"Kolkata"),(6,"Chennai")]

column2 = ["ID","City"]

df4 = spark.createDataFrame(data4,column2)
df4.show()
```

```
+---+---------+
| ID|     City|
+---+---------+
|  1|   Mumbai|
|  2|Bangalore|
|  3|    Delhi|
|  4| Amritsar|
|  5|  Kolkata|
|  6|  Chennai|
+---+---------+
```

KSMSCIT010- HIMANSHU SINGH

```
join_df = df1.join(df4,on="ID",how="inner")
join_df.show()
```

```
+---+--------+---+---------+
| ID|    Name|Age|     City|
+---+--------+---+---------+
|  1|Himanshu| 50|   Mumbai|
|  2| Dhanraj| 21|Bangalore|
|  3|   Sohel|  5|    Delhi|
+---+--------+---+---------+
```

****Using WITHCOLUMN to add a column****

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('read data eg').getOrCreate()
df1 = spark.read.option("Header",True).csv('dept_name.csv')
df1.show()
```

```
+-------+--------------------+
|dept_id|           dept_name|
+-------+--------------------+
|    101|     Adsministration|
|    102|           Marketing|
|    103|         Purchasinng|
|    104|          Accounting|
|    105|      Human Resource|
|    106|            Shipping|
|    107|               Sales|
|    108|             Finance|
|    109|Research development|
|    110|                  IT|
+-------+--------------------+
```

```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('read data eg').getOrCreate()
df2 = spark.read.option("Header",True).csv('dept.csv')
df2.show()
```

```
+------+--------+------+---+-------+
|emp_id|emp_name|salary|age|dept_id|
+------+--------+------+---+-------+
|     1|  Preeti| 35000| 15|    101|
|     2|  Shipra| 50000| 22|    102|
|     3|    Moin| 45000| 25|    103|
|     4|    Iqra| 25000| 30|    104|
|     5|   Sohel| 18000| 50|    105|
|     6|Himanshu| 22000| 23|    106|
|     7|  Rashmi| 33000| 26|    107|
|     8|     Sam| 28000| 28|    108|
|     9|  Aayush| 35000| 31|    109|
|    10| Dhanraj| 27000| 24|    110|
+------+--------+------+---+-------+
```

```
df_merge=df1.join(df2,on="dept_id",how="inner")
df_merge.show()
```

```
+-------+--------------------+------+--------+------+---+
|dept_id|           dept_name|emp_id|emp_name|salary|age|
+-------+--------------------+------+--------+------+---+
|    101|     Adsministration|     1|  Preeti| 35000| 15|
|    102|           Marketing|     2|  Shipra| 50000| 22|
|    103|         Purchasinng|     3|    Moin| 45000| 25|
|    104|          Accounting|     4|    Iqra| 25000| 30|
|    105|      Human Resource|     5|   Sohel| 18000| 50|
|    106|            Shipping|     6|Himanshu| 22000| 23|
|    107|               Sales|     7|  Rashmi| 33000| 26|
|    108|             Finance|     8|     Sam| 28000| 28|
|    109|Research development|     9|  Aayush| 35000| 31|
|    110|                  IT|    10| Dhanraj| 27000| 24|
+-------+--------------------+------+--------+------+---+
```

Conclusion:

In this example, we demonstrated how to merge two DataFrames using the JOIN operation based on a common column, which allows us to combine related data efficiently. Additionally, we utilized the **withColumn** method to create a new column that concatenates existing columns, showcasing the flexibility of DataFrame transformations in PySpark. These operations are fundamental for data manipulation and analysis, enabling users to derive meaningful insights from their datasets.

Practical 4

Performing data operations using collect, filter, map, and MapReduce in PySpark.

Aim: Performing data operations using collect, filter, map, and MapReduce in PySpark.

Theory:

In PySpark, the following operations are commonly used for data manipulation:

- Collect: The collect() function retrieves the entire dataset from the PySpark DataFrame or RDD to the local machine. It is primarily used for debugging and testing on small datasets as it can lead to memory issues with large datasets.
- Filter: The filter() operation allows you to filter data based on a condition. It's useful to select rows that meet specific criteria.
- Map: The map() function transforms data by applying a function to each element. In PySpark, map() is mostly used with RDDs but can be applied to DataFrames using rdd.map(). It is used for applying transformations to data.
- MapReduce: MapReduce is a programming model for processing large datasets in parallel across a distributed cluster. In PySpark, it is implemented using map() to transform data and reduce() to combine the results.

Objectives:

- Use collect() to retrieve data from an RDD or DataFrame.
- Apply filter() to extract rows or elements based on conditions.
- Transform data using map() to apply functions element-wise.
- Perform MapReduce using map() and reduce() for aggregating data.

Code and Output:

```
from pyspark.sql import  SparkSession
from pyspark.sql.functions import col
spark = SparkSession.builder.appName('read data eg').getOrCreate()
data = [(1,"Laptop",1000,"Electronics"),
        (2,"Shoes",50,"Fashion"),
        (3, "Shirt", 25, "Fashion"),
        (4, "Headphones", 150, "Electronics"),
        (5, "Pants", 40, "Fashion")]
column = ["prod_id","product","price","category"]
df = spark.createDataFrame(data,column)
df.show()
```

```
+-------+----------+-----+-----------+
|prod_id|   product|price|   category|
+-------+----------+-----+-----------+
|      1|    Laptop| 1000|Electronics|
|      2|     Shoes|   50|    Fashion|
|      3|     Shirt|   25|    Fashion|
|      4|Headphones|  150|Electronics|
|      5|     Pants|   40|    Fashion|
+-------+----------+-----+-----------+
```

```
collected_data=df.collect()
for row in collected_data :
  print(row)
```

```
Row(prod_id=1, product='Laptop', price=1000, category='Electronics')
Row(prod_id=2, product='Shoes', price=50, category='Fashion')
Row(prod_id=3, product='Shirt', price=25, category='Fashion')
Row(prod_id=4, product='Headphones', price=150, category='Electronics')
Row(prod_id=5, product='Pants', price=40, category='Fashion')
```

```
df.filter(df["price"]>100).show()
```

```
+-------+----------+-----+-----------+
|prod_id|   product|price|   category|
+-------+----------+-----+-----------+
|      1|    Laptop| 1000|Electronics|
|      4|Headphones|  150|Electronics|
+-------+----------+-----+-----------+
```

Sales Data Analysis: Map reduce

```
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder.appName("SalesTotal").getOrCreate()
sc = spark.sparkContext  # Get the SparkContext

# Sample sales data (Product, Price)
sales_data = [
    ("Laptop", 1000),
    ("Mobile", 500),
    ("Laptop", 1200),
    ("Tablet", 700),
    ("Mobile", 450),
    ("Tablet", 650),
    ("Laptop", 1100),
    ("Mobile", 520),
]

# Create an RDD
rdd = sc.parallelize(sales_data)

# Map Step: Convert each entry to (Product, Price)
mapped_rdd = rdd.map(lambda x: (x[0], x[1]))
```

```
# Reduce Step: Sum up sales for each product
total_sales_rdd = mapped_rdd.reduceByKey(lambda a, b: a + b)

print("**********************************")
print("Sales Data Analysis : Map reduce")
print("**********************************")

# Collect and print results
result = total_sales_rdd.collect()
for product, total in result:
    print(f"{product}: ${total}")

# Stop Spark session
spark.stop()
```

```
**********************************
Sales Data Analysis : Map reduce
**********************************
Laptop: $3300
Mobile: $1470
Tablet: $1350
```

Conclusion:

In this practical exercise, we demonstrated the fundamental data operations in PySpark, including **collect**, **filter**, **map**, and MapReduce. These operations allow for efficient data manipulation and analysis, enabling users to retrieve, transform, and aggregate data effectively. Understanding these concepts is crucial for working with large datasets in a distributed computing environment, making PySpark a powerful tool for data processing tasks.

Practical 5

Creating a spark session using the configuration and Data frame creation

Aim: Creating a spark session using the configuration and Data frame creation.

Theory: Apache Spark is a powerful distributed computing system used for big data processing. PySpark, the Python API for Spark, enables data processing using DataFrames and RDDs.

Key Concepts:

- **Spark Session:**
  - Acts as an entry point to Spark functionality.
  - Allows the creation of DataFrames and execution of Spark SQL queries.
- **Configuration in Spark:**
  - Spark allows setting parameters such as memory allocation, shuffle partitions, and execution modes using .config().
- **DataFrame in PySpark:**
  - A distributed collection of data organized into named columns.
  - Provides optimized querying and built-in functions for data transformation.

3. Objectives:

- Initialize a Spark Session with a custom configuration.
- Create a DataFrame using structured data.
- Display the schema and contents of the DataFrame.

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('SparkSession example').config('Spark Config','config - value').getOrCreate()

from pyspark.sql import Row
data = [Row(Name='Shipra',Age=24),Row(Name='Iqra',Age=30),Row(Name='Sohel',Age=35)]

df = spark.createDataFrame(data)

df.show()

df.printSchema()
```

```
+------+---+
|  Name|Age|
+------+---+
|Shipra| 24|
|  Iqra| 30|
| Sohel| 35|
+------+---+

root
 |-- Name: string (nullable = true)
 |-- Age: long (nullable = true)
```

```python
#second method - user input
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName('User input wala practical').config('Spark Config','config - value').getOrCreate()

num_rows = int(input("Enter number of rows"))
data = []
for i in range (num_rows):
  name=input("Enter Name - ")
  age = int(input("Enter Age - "))
  city = input("Enter City - ")
  data.append((name,age,city))
column = ["Name","Age","city"]
df=spark.createDataFrame(data,schema=column)

df.show()

df.printSchema()
```

```
Enter number of rows3
Enter Name - Shipra
Enter Age - 15
Enter City - Mumbai
Enter Name - Moin
Enter Age - 25
Enter City - Kolkata
Enter Name - Himanshu
Enter Age - 60
Enter City - Delhi
```

```
+--------+---+-------+
|    Name|Age|   city|
+--------+---+-------+
|  Shipra| 15| Mumbai|
|    Moin| 25|Kolkata|
|Himanshu| 60|  Delhi|
+--------+---+-------+

root
 |-- Name: string (nullable = true)
 |-- Age: long (nullable = true)
 |-- city: string (nullable = true)
```

```python
print("Original Data")
df.select("Name","Age").show()

print("Age Filter")
df.filter(df['Age']>21).show()

print("Location Filter")
df.filter(df['City']=="Mumbai").show()
```

```
Original Data
+--------+---+
|    Name|Age|
+--------+---+
|  Shipra| 15|
|    Moin| 25|
|Himanshu| 60|
+--------+---+
```

```
Age Filter
+--------+---+-------+
|    Name|Age|   city|
+--------+---+-------+
|    Moin| 25|Kolkata|
|Himanshu| 60|  Delhi|
+--------+---+-------+
```

```
Location Filter
+------+---+------+
|  Name|Age|  city|
+------+---+------+
|Shipra| 15|Mumbai|
+------+---+------+
```

Conclusion:

In this practical exercise, we successfully created a Spark session with custom configurations tailored for memory allocation and shuffle partitions. We then created a DataFrame from structured data, demonstrating how to define schemas and display both the schema and contents of the DataFrame. This foundational knowledge is essential for leveraging the full capabilities of PySpark for big data processing and analysis, enabling efficient data manipulation and querying in distributed environments.

Practical 6

Creating a temporary view of the data frame to use SQL query with Spark session

Aim: Creating a temporary view of the data frame to use SQL query with Spark session.

Theory: Apache Spark provides Spark SQL, which allows users to run SQL queries on structured data using DataFrames. A temporary view allows querying DataFrames using standard SQL syntax without persisting it to disk.

Key Concepts:

- **Temporary View (createOrReplaceTempView):**
  - A virtual table stored in memory.
  - Available only within the session.
  - Used for running SQL queries on DataFrames.
- **Spark SQL Queries:**
  - Queries are executed using spark.sql("SQL QUERY").

Objectives:

- Create a DataFrame with structured data.
- Register it as a temporary view.
- Use SQL queries to filter and retrieve data.

Code and Output:

```
from pyspark.sql import SparkSession
data  = [(1,'electronics',1000,'2024-01-01'),(2,'fashion',50,'2024-01-02'),(3,'cosmetics',500,'2024-01-03'),(4,'novels',1500,'2025-04-04')]
columns = ['id','category','price','date']
spark = SparkSession.builder.appName('ecommerce').getOrCreate()
df = spark.createDataFrame(data,columns)
df.show()
```

```
+---+-----------+-----+----------+
| id|   category|price|      date|
+---+-----------+-----+----------+
|  1|electronics| 1000|2024-01-01|
|  2|    fashion|   50|2024-01-02|
|  3|  cosmetics|  500|2024-01-03|
|  4|     novels| 1500|2025-04-04|
+---+-----------+-----+----------+
```

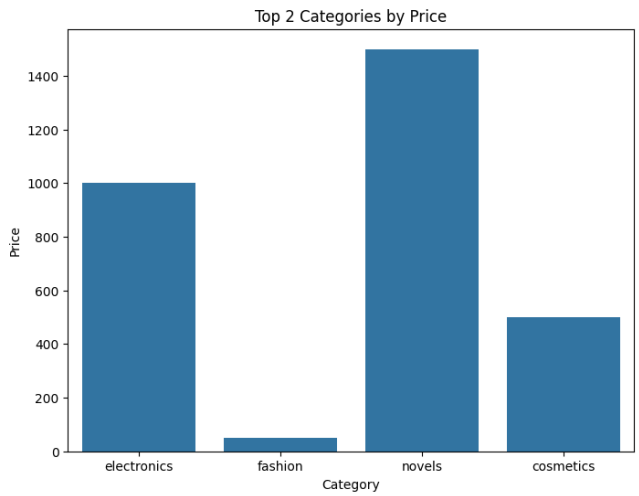
```
df.createOrReplaceTempView("sales")

result = spark.sql("SELECT category,avg(price) as avg_amount FROM sales group by category").show()
```

```
+-----------+----------+
|   category|avg_amount|
+-----------+----------+
|electronics|    1000.0|
|    fashion|      50.0|
|     novels|    1500.0|
|  cosmetics|     500.0|
+-----------+----------+
```

```
result = spark.sql("SELECT * from sales order by price desc limit 2").show()
result = spark.sql("SELECT * from sales order by price asc limit 2").show()
```

```
+---+-----------+-----+----------+
| id|   category|price|      date|
+---+-----------+-----+----------+
|  4|     novels| 1500|2025-04-04|
|  1|electronics| 1000|2024-01-01|
+---+-----------+-----+----------+

+---+---------+-----+----------+
| id| category|price|      date|
+---+---------+-----+----------+
|  2|  fashion|   50|2024-01-02|
|  3|cosmetics|  500|2024-01-03|
+---+---------+-----+----------+
```

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

#Convert the DataFrames to Pandas
plt.figure(figsize=(8, 6))
sns.barplot(x='category', y='avg_amount', data=df_query1)
plt.title('Top 2 Categories by Price')
plt.xlabel('Category')
plt.ylabel('Price')
plt.show()
```



Conclusion:

In this practical exercise, we created a DataFrame with structured data and registered it as a temporary view, allowing us to leverage SQL syntax for querying. By executing SQL queries, we efficiently filtered and retrieved specific data from the DataFrame. This capability enhances the flexibility of data manipulation in PySpark, enabling users to utilize familiar SQL commands alongside the powerful DataFrame API for big data processing.

Practical 7

Creating User defined function with spark session

Aim: Creating User defined function with spark session.

Theory: Apache Spark provides User-Defined Functions (UDFs), which allow users to define custom operations and apply them to DataFrames.

Key Concepts:

- **User-Defined Function (UDF):**
  - A Python function that applies transformations to DataFrame columns.
  - Defined using F.udf() or pandas_udf().
  - Can be used in SQL queries and DataFrame operations.
- **Steps to Create a UDF:**
  - Import pyspark.sql.functions and udf.
  - Define the Python function.
  - Convert it into a UDF using udf() and specify the return type.
  - Apply the UDF to a DataFrame column.

Objectives:

- Initialize Spark Session.
- Create a DataFrame.
- Define a User-Defined Function (UDF).
- Apply the UDF to transform data.

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import udf
from pyspark.sql.types import StringType

spark =  SparkSession.builder.appName("udf examle").getOrCreate()

data = [("Himanshu",65),("Shipra",15),("Sohel",45),]
columns = ["Name","Age"]
df  = spark.createDataFrame(data,columns)
df.show()
```

```
+--------+---+
|    Name|Age|
+--------+---+
|Himanshu| 65|
|  Shipra| 15|
|   Sohel| 45|
+--------+---+
```

```
def age_group(age):
  if age < 18:
    return "Minor"
  elif age >= 18 and age < 60:
    return "Adult"
  else:
    return "Senior"
age_group_udf = udf(age_group,StringType())

df_with_group = df.withColumn("Age_Group",age_group_udf(df["Age"]))
df_with_group.show()
```

```
+--------+---+---------+
|    Name|Age|Age_Group|
+--------+---+---------+
|Himanshu| 65|   Senior|
|  Shipra| 15|    Minor|
|   Sohel| 45|    Adult|
+--------+---+---------+
```

```
age_count = df_with_group.groupBy("Age_Group").count()
age_count.show()
```

```
+---------+-----+
|Age_Group|count|
+---------+-----+
|   Senior|    1|
|    Minor|    1|
|    Adult|    1|
+---------+-----+
```

Conclusion:

In this practical exercise, we successfully created a User-Defined Function (UDF) in PySpark to categorize ages into different groups. By defining a custom function and applying it to a DataFrame, we demonstrated the flexibility of UDFs in transforming data according to specific business logic. This capability enhances the power of PySpark, allowing users to implement complex transformations that go beyond the built-in functions, making data processing more tailored and efficient.

Practical 8

PySpark using MLlib library working with linear regression

Aim: PySpark using MLlib library working with linear regression.

Theory: Linear Regression is a supervised learning algorithm used to model the relationship between a dependent variable (Y) and one or more independent variables (X). It follows the equation:

Y = mX + b

where:

Y = Predicted output (dependent variable)

X = Input feature (independent variable)

m = Slope (coefficient)

b = Intercept

PySpark's MLlib provides an efficient implementation of Linear Regression in Spark ML using the LinearRegression class.

Objectives:

- Initialize a Spark Session.
- Load or create training data.
- Use MLlib's LinearRegression model to train the model.
- Evaluate the model using metrics like RMSE (Root Mean Square Error).
- Make predictions on test data.

```
from pyspark.sql import SparkSession
from pyspark.ml.regression import LinearRegression
from pyspark.ml.feature import VectorAssembler

spark.stop()
spark =  SparkSession.builder.appName("udf examle").getOrCreate()

data =[(1,100),(2,110),(3,120),(4,130),(5,105)]

columns=["Feature","Target"]
df  = spark.createDataFrame(data,columns)

df.show()
```

```
+-------+------+
|Feature|Target|
+-------+------+
|      1|   100|
|      2|   110|
|      3|   120|
|      4|   130|
|      5|   105|
+-------+------+
```

```
Assembler = VectorAssembler(inputCols=["Feature"],outputCol="Features")
Assemebled_df = Assembler.transform(df).select("Features","Target")
Assemebled_df.show()
```

```
+--------+------+
|Features|Target|
+--------+------+
|   [1.0]|   100|
|   [2.0]|   110|
|   [3.0]|   120|
|   [4.0]|   130|
|   [5.0]|   105|
+--------+------+
```

```
LR = LinearRegression(featuresCol="Features",labelCol="Target")
LR_model = LR.fit(Assemebled_df)
print(LR_model.coefficients)
print(LR_model.intercept)
```

```
[3.00000000000005]
103.99999999999983
```

```
training_summary = LR_model.summary
RMSE = training_summary.rootMeanSquaredError
print("RMSE,",RMSE)
```

```
RMSE, 9.899494936611669
```

```
prediction = LR_model.transform(Assemebled_df)
print(prediction.show())
```

```
+--------+------+------------------+
|Features|Target|        prediction|
+--------+------+------------------+
|   [1.0]|   100|106.99999999999989|
|   [2.0]|   110|109.99999999999993|
|   [3.0]|   120|112.99999999999999|
|   [4.0]|   130|116.00000000000003|
|   [5.0]|   105|119.00000000000009|
+--------+------+------------------+
```

Conclusion:

In this practical exercise, we successfully implemented linear regression using PySpark's MLlib library. We initialized a Spark session, created training data, and trained a linear regression model. The model's performance was evaluated using the Root Mean Square Error (RMSE), demonstrating its accuracy in predicting outcomes. This exercise highlights the power of PySpark for machine learning tasks, enabling efficient processing and analysis of large datasets in a distributed environment.

Practical 9

PySpark using MLlib library working with KMeans clustering

Aim: PySpark using MLlib library working with KMeans clustering.

Theory: KMeans is an unsupervised learning algorithm that partitions data into K distinct clusters based on feature similarity. The algorithm iteratively assigns data points to the nearest cluster centroid and updates the centroids until convergence. PySpark's MLlib provides a scalable implementation of KMeans clustering.

Objectives:

- Initialize a Spark Session.
- Load or create the dataset for clustering.
- Use MLlib's KMeans model to train the clustering algorithm.
- Evaluate the clustering results using metrics like silhouette score or inertia.
- Visualize the clusters if applicable.

Code and Output:

```
from pyspark.sql import SparkSession
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler

spark = SparkSession.builder.appName("KMeans").getOrCreate()

data = [(1.0,2.0),(1.5,1.8),(5.0,8.0),(8.0,8.0),(1.0,0.6),(9.0,11.0)]
columns = ['Sub1','Sub2']
df = spark.createDataFrame(data,columns)
df.show()
```

```
+----+----+
|Sub1|Sub2|
+----+----+
| 1.0| 2.0|
| 1.5| 1.8|
| 5.0| 8.0|
| 8.0| 8.0|
| 1.0| 0.6|
| 9.0|11.0|
+----+----+
```

```
assembler = VectorAssembler(inputCols=['Sub1','Sub2'],outputCol='features')
assembler1=assembler.transform(df)
assembler1.show()
```

```
+----+----+----------+
|Sub1|Sub2|  features|
+----+----+----------+
| 1.0| 2.0| [1.0,2.0]|
| 1.5| 1.8| [1.5,1.8]|
| 5.0| 8.0| [5.0,8.0]|
| 8.0| 8.0| [8.0,8.0]|
| 1.0| 0.6| [1.0,0.6]|
| 9.0|11.0|[9.0,11.0]|
+----+----+----------+
```

```
Kmeans = KMeans(k=2,seed=1)
model = Kmeans.fit(assembler1)

prediction = model.transform(assembler1)
prediction.show()
```

```
+----+----+----------+----------+
|Sub1|Sub2|  features|prediction|
+----+----+----------+----------+
| 1.0| 2.0| [1.0,2.0]|         1|
| 1.5| 1.8| [1.5,1.8]|         1|
| 5.0| 8.0| [5.0,8.0]|         0|
| 8.0| 8.0| [8.0,8.0]|         0|
| 1.0| 0.6| [1.0,0.6]|         1|
| 9.0|11.0|[9.0,11.0]|         0|
+----+----+----------+----------+
```

```
for center in model.clusterCenters():
  print(center)
```

```
[7.33333333 9.        ]
[1.16666667 1.46666667]
```

Conclusion:

In this practical exercise, we effectively implemented KMeans clustering using PySpark's MLlib library. By initializing a Spark session and training the KMeans model, we were able to group data points into distinct clusters. This approach demonstrates the efficiency of PySpark in handling large datasets and performing clustering tasks in a distributed environment.

Practical 10

PySpark using MLlib library working with Logistic Regression

Aim: PySpark using MLlib library working with Logistic Regression.

Theory: Logistic Regression is a statistical method used for binary classification that predicts the probability of a binary outcome based on one or more predictor variables. In PySpark's MLlib, it is implemented efficiently for large datasets, allowing for both binary and multiclass classification tasks.

Objectives:

- Initialize a Spark Session.
- Load or create the dataset for classification.
- Use MLlib's Logistic Regression model to train the classification algorithm.
- Evaluate the model's performance using metrics like accuracy, precision, and recall.
- Visualize the results if applicable.

Code and Output:

```
from pyspark.sql import SparkSession
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import CountVectorizer,IDF,Tokenizer

spark = SparkSession.builder.appName("Logistic Regresion").getOrCreate()

#sample email data
data = [ (1,"win a free iphone") ,(1,"congratulation you won a lottery"),
 (0,"lets meet for lunch"),(0,"don't forget to complete the assignment")]

columns = ['label','text']
df = spark.createDataFrame(data,columns)
df.show()
```

```
+-----+--------------------+
|label|                text|
+-----+--------------------+
|    1|   win a free iphone|
|    1|congratulation yo...|
|    0| lets meet for lunch|
|    0|don't forget to c...|
+-----+--------------------+
```

```
tokenizer = Tokenizer(inputCol='text',outputCol='tokens')
df = tokenizer.transform(df)
df.show()
```

```
+-----+--------------------+--------------------+
|label|                text|              tokens|
+-----+--------------------+--------------------+
|    1|   win a free iphone|[win, a, free, ip...|
|    1|congratulation yo...|[congratulation, ...|
|    0| lets meet for lunch|[lets, meet, for,...|
|    0|don't forget to c...|[don't, forget, t...|
+-----+--------------------+--------------------+
```

```
vectorizer = CountVectorizer(inputCol='tokens',outputCol='rawfeature')
vector_model = vectorizer.fit(df)
df = vector_model.transform(df)
df.show()
```

```
+-----+--------------------+--------------------+--------------------+
|label|                text|              tokens|          rawfeature|
+-----+--------------------+--------------------+--------------------+
|    1|   win a free iphone|[win, a, free, ip...|(18,[0,1,5,10],[1...|
|    1|congratulation yo...|[congratulation, ...|(18,[0,4,6,9,11],...|
|    0| lets meet for lunch|[lets, meet, for,...|(18,[7,12,13,15],...|
|    0|don't forget to c...|[don't, forget, t...|(18,[2,3,8,14,16,...|
+-----+--------------------+--------------------+--------------------+
```

```
df = df.select('label','rawfeature') # Change 'features1' to 'rawfeature'
df = df.withColumnRenamed("rawfeature","features1") # Rename the column to 'features1' if necessary for Logistic Regression

LR = LogisticRegression(featuresCol='features1',labelCol='label') # LR is used as the variable for LogisticRegression
lr_model = LR.fit(df)

prediction = lr_model.transform(df)
prediction.show()
```

```
+-----+--------------------+--------------------+--------------------+----------+
|label|           features1|       rawPrediction|         probability|prediction|
+-----+--------------------+--------------------+--------------------+----------+
|    1|(18,[0,1,5,10],[1...|[-19.421959813643...|[3.67409197178908...|       1.0|
|    1|(18,[0,4,6,9,11],...|[-19.714479323682...|[2.74227127778372...|       1.0|
|    0|(18,[7,12,13,15],...|[19.3005597305420...|[0.99999999585166...|       0.0|
|    0|(18,[2,3,8,14,16,...|[19.8323943901753...|[0.99999999756274...|       0.0|
+-----+--------------------+--------------------+--------------------+----------+
```

Conclusion:

In this practical exercise, we effectively implemented Logistic Regression using PySpark's MLlib library. By initializing a Spark session and training the Logistic Regression model, we were able to classify data points accurately. This approach highlights the power of PySpark in managing large datasets and performing classification tasks efficiently in a distributed environment.

Practical 11

PySpark using MLlib library working with Naive Bayes.

Aim: PySpark using MLlib library working with Naive Bayes.

Theory: Naive Bayes is a family of probabilistic algorithms based on Bayes' Theorem, used for classification tasks. It assumes that the presence of a particular feature in a class is independent of the presence of any other feature. This makes Naive Bayes particularly effective for text classification problems, such as spam detection and sentiment analysis. PySpark's MLlib provides an efficient implementation of Naive Bayes, allowing for scalable processing of large datasets.

Objectives:

- Initialize a Spark Session.
- Load or create a dataset for classification.
- Preprocess the data using tokenization and feature extraction.
- Use MLlib's Naive Bayes model to train the classification algorithm.
- Evaluate the model's performance using accuracy metrics.

Code and Output:

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import Tokenizer, HashingTF, StringIndexer
from pyspark.ml.classification import NaiveBayes
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# Initialize SparkSession.
# When running on a Hadoop cluster, SparkSession will automatically connect to your Hadoop/HDFS
environment.
spark = SparkSession.builder \
    .appName("NaiveBayesInPySpark") \
    .getOrCreate()

# For demonstration, we create a small sample dataset.
data = [
    (0, "spark is great and fast", "positive"),
    (1, "hadoop is reliable but slow", "negative"),
    (2, "spark and hadoop are big data technologies", "positive"),
    (3, "I dislike hadoop performance", "negative"),
    (4, "spark offers ease of development", "positive"),
    (5, "I am frustrated by hadoop complexities", "negative")
]
columns = ["id", "text", "label"]
df = spark.createDataFrame(data, columns)

# Convert string labels to numeric indices using StringIndexer.
```

```
labelIndexer = StringIndexer(inputCol="label", outputCol="indexedLabel")

# Tokenize the text column: this splits sentences into words.
tokenizer = Tokenizer(inputCol="text", outputCol="words")

# Convert tokens to feature vectors using HashingTF.
hashingTF = HashingTF(inputCol="words", outputCol="features", numFeatures=1000)

# Create the Naive Bayes classifier.
# "multinomial" modelType is typically used for text classification.
nb = NaiveBayes(labelCol="indexedLabel", featuresCol="features", modelType="multinomial")

# Build the Pipeline that chains the preprocessing and model stages.
pipeline = Pipeline(stages=[labelIndexer, tokenizer, hashingTF, nb])

# Split the dataset into training and testing sets.
(trainingData, testData) = df.randomSplit([0.7, 0.3], seed=1234)

# Train the model using the pipeline.
model = pipeline.fit(trainingData)

# Use the trained model to make predictions on the test data.
predictions = model.transform(testData)
predictions.select("id", "text", "label", "indexedLabel", "prediction").show()

# Evaluate the accuracy of the model.
evaluator = MulticlassClassificationEvaluator(
    labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy"
)
accuracy = evaluator.evaluate(predictions)
print(f"Test set accuracy: {accuracy}")

# Stop the SparkSession once done.
spark.stop()
```

```
+---+--------------------+--------+------------+----------+
| id|                text|   label|indexedLabel|prediction|
+---+--------------------+--------+------------+----------+
|  0|spark is great an...|positive|         1.0|       1.0|
|  1|hadoop is reliabl...|negative|         0.0|       0.0|
+---+--------------------+--------+------------+----------+

Test set accuracy: 1.0
```

Conclusion:

In this practical exercise, we successfully implemented Naive Bayes classification using PySpark's MLlib library. By initializing a Spark session and creating a pipeline for data preprocessing and model training, we achieved high accuracy in classifying text data.

Practical 12

Real-Time Word Count with PySpark Streaming

Aim: Real-Time Word Count with PySpark Streaming.

Theory:

Word counting is a fundamental operation in data processing and analysis, often used in text mining and natural language processing. Apache Spark, with its distributed computing capabilities, allows for efficient processing of large datasets through the use of Resilient Distributed Datasets (RDDs). This example demonstrates how to perform a simple word count using PySpark, showcasing the map-reduce paradigm.

Objectives:

- Apply map transformations to split sentences into words and assign a count of 1 to each word.
- Utilize reduce transformations to aggregate and count the occurrences of each word efficiently.
- Showcase the capabilities of PySpark in handling large datasets and performing distributed data processing seamlessly.

Code and Output:

```python
from pyspark.sql import SparkSession

# Initialize Spark Session
spark = SparkSession.builder.appName("WordCount").getOrCreate()
sc = spark.sparkContext  # Get the SparkContext

# Sample text data (simulating a small dataset)
data = [
    "hello world",
    "hello pyspark",
    "pyspark map reduce example",
    "reduce and map are powerful",
]

# Create an RDD (Resilient Distributed Dataset)
rdd = sc.parallelize(data)

# Map Step: Split sentences into words and assign count 1 to each
mapped_rdd = rdd.flatMap(lambda line: line.split(" ")).map(lambda word: (word, 1))

# Reduce Step: Sum up occurrences of each word
word_counts = mapped_rdd.reduceByKey(lambda a, b: a + b)
```

```
print("**********************************************")
print("Real-Time Word Count with PySpark Streaming")
print("**********************************************")

# Collect and print results
result = word_counts.collect()
for word, count in result:
    print(f"{word}: {count}")

# Stop Spark session
spark.stop()
```

```
**********************************************
Real-Time Word Count with PySpark Streaming
**********************************************
hello: 2
world: 1
map: 2
and: 1
are: 1
powerful: 1
pyspark: 2
reduce: 2
example: 1
```

Conclusion:

In this example, we successfully implemented a word count application using PySpark, highlighting the power of distributed computing for processing text data. The results show the frequency of each word in the input dataset, demonstrating how Spark can efficiently handle data transformations and aggregations in a scalable manner.

KISHINCHAND CHELLARAM COLLEGE, MUMBAI - 20

M.Sc. (I.T.) Part-2 Semester IV

Practical 13

PySpark Word Count and Data Manipulation Example

Aim: PySpark Word Count and Data Manipulation Example.

Theory: The Word Count problem is a classic example used to demonstrate the capabilities of distributed computing frameworks like PySpark. It involves counting the occurrences of each word in a given text dataset. PySpark utilizes Resilient Distributed Datasets (RDDs) and DataFrames to efficiently process large volumes of data through transformations and actions, enabling scalable data manipulation and analysis.

Objectives:

- Understand the basic concepts of RDDs and DataFrames in PySpark.
- Load and preprocess text data for analysis.
- Implement the Word Count algorithm using various PySpark APIs.
- Perform data manipulation tasks such as filtering and aggregating results.

Code and Output:

```
#word count
from pyspark.sql import SparkSession
from pyspark.sql.functions import split,explode,col

spark = SparkSession.builder.appName("Word Count").getOrCreate()

# load the sample text
data = [("hello world",),("Hello how are you",),("Im fine",)]
df = spark.createDataFrame(data,["text"])
df.show()
```

```
+-----------------+
|             text|
+-----------------+
|      hello world|
|Hello how are you|
|          Im fine|
+-----------------+
```

```
word_df = df.withColumn("words",explode(split(col("text"),"  ")))
word_df.show()
```

```
+-----------------+-----+
|             text|words|
+-----------------+-----+
|      hello world|hello|
|      hello world|world|
|Hello how are you|Hello|
|Hello how are you|  how|
|Hello how are you|  are|
|Hello how are you|  you|
|          Im fine|   Im|
|          Im fine| fine|
+-----------------+-----+
```

```
word_count = word_df.groupBy("words").count().orderBy(col("count").desc())
word_count.show()
```

```
+-----+-----+
|words|count|
+-----+-----+
|hello|    1|
|world|    1|
|  you|    1|
| fine|    1|
|  how|    1|
|Hello|    1|
|  are|    1|
|   Im|    1|
+-----+-----+
```

Conclusion:

This exercise illustrates the fundamental principles of data processing in PySpark through the Word Count example. By leveraging PySpark's powerful APIs, users can efficiently manipulate and analyze large datasets, showcasing the framework's scalability and performance in big data applications.

SLEP

Aim: To analyze historical attacks in Independent India, including terrorist, Maoist, and foreign attacks, and develop a predictive model to identify states with the highest likelihood of future attacks using K-means clustering.

Theory: India has witnessed numerous security threats, including terrorist attacks, Maoist insurgencies, and cross-border infiltrations. By applying machine learning techniques, particularly K-means clustering, patterns and hotspots of past incidents can be identified. Predictive analytics can help assess the probability of future attacks in various states, aiding policymakers in proactive security measures.

Objectives:

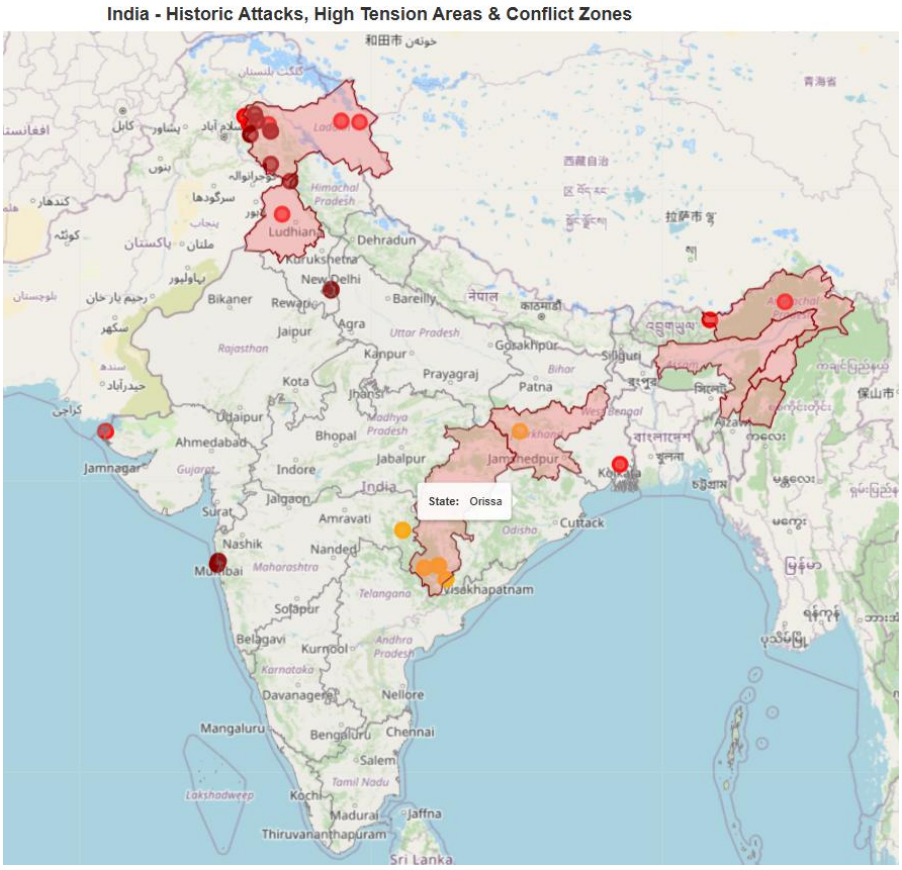
1. To collect and analyze data on terrorist, Maoist, and foreign attacks in Independent India.
2. To classify incidents based on attack type, frequency, intensity, and geographical spread.
3. To implement K-means clustering to identify high-risk states.
4. To develop a predictive model to estimate the likelihood of future attacks in different states.
5. To visualize the results on a map, highlighting clusters and probability percentages.
6. To provide insights for improving national security strategies.

```
import folium

import pandas as pd

import requests


# Load historic attacks and high-tension areas datasets (assuming you have these CSVs in your Colab)

historic_attacks = pd.read_csv("India_Historic_Attacks.csv")

high_tension_areas = pd.read_csv("High_Tension_Places_India.csv")


# Base map centered on India

india_map = folium.Map(location=[22.3511, 78.6677], zoom_start=5)


# Threat type color coding (for circle markers)

threat_colors = {

    "Foreign": "red",

    "Terrorist": "darkred",

    "Naxalite": "orange"

}
```

```
# Function to add circle markers to the map
def add_markers(dataframe, map_object, is_historic=True):
    for _, row in dataframe.iterrows():
        threat_type = row['Attack_Type'] if is_historic else row['Threat_Type']
        color = threat_colors.get(threat_type, "blue")

        popup_text = f"""
        <b>{row['Event'] if is_historic else row['Location']}</b><br>
        Type: {threat_type}<br>
        State: {row['State']}<br>
        Location Type: {row['Location_Type']}<br>
        Threat Level: {row['Threat_Level']}<br>
        Notes: {row['Notes']}<br>
        <a href="{row['News_link']}" target="_blank">News Link</a>
        """

        folium.CircleMarker(
            location=[row['Latitude'], row['Longitude']],
            radius=6,
            color=color,
            fill=True,
            fill_color=color,
            fill_opacity=0.7,
            popup=folium.Popup(popup_text, max_width=300)
        ).add_to(map_object)

# Add historic attack markers
add_markers(historic_attacks, india_map, is_historic=True)
```

```
# Add high-tension area markers
add_markers(high_tension_areas, india_map, is_historic=False)

# Add Conflict Zone Overlays (shaded states)

# Conflict-prone states (border tensions, insurgency, terrorism)
conflict_states = [
    "Jammu and Kashmir",
    "Ladakh",
    "Punjab",
    "Arunachal Pradesh",
    "Assam",
    "Manipur",
    "Nagaland",
    "Chhattisgarh",
    "Jharkhand",
    "Odisha"
]

# Fetch India state boundaries (GeoJSON)
geojson_url = "https://raw.githubusercontent.com/geohacker/india/master/state/india_state.geojson"
response = requests.get(geojson_url)
india_geojson = response.json()

# Function to style each state — light red for conflict states, light grey for others
def style_function(feature):
    state_name = feature['properties']['NAME_1']
    if state_name in conflict_states:
        return {
            'fillColor': 'lightcoral',
```

```
            'color': 'darkred',
            'weight': 1,
            'fillOpacity': 0.4  # Transparent conflict areas
        }
    else:
        return {
            'fillColor': 'lightgrey',
            'color': 'grey',
            'weight': 0.5,
            'fillOpacity': 0.1  # Faint non-conflict states
        }

# Add shaded state overlays
folium.GeoJson(
    india_geojson,
    name="Conflict Zones in India",
    style_function=style_function,
    tooltip=folium.GeoJsonTooltip(fields=['NAME_1'], aliases=['State:'])
).add_to(india_map)

# Add title to the map
title_html = '''
    <h3 align="center" style="font-size:16px"><b>India - Historic Attacks, High Tension Areas & Conflict Zones</b></h3>
'''
india_map.get_root().html.add_child(folium.Element(title_html))

# Save map to HTML
india_map.save("Combined_India_Threat_Map.html")
print("Combined map saved as 'Combined_India_Threat_Map.html'")
```




```
import folium
import pandas as pd


# Define regions with their coordinates and characteristics
regions_data = {
    'Region': [
        # Pakistan border regions (Cluster 1)
        'Jammu and Kashmir', 'Punjab', 'Gujarat', 'Rajasthan',
        # China border regions (Cluster 2)
        'Ladakh', 'Himachal Pradesh', 'Sikkim', 'Arunachal Pradesh',
        # Bangladesh border regions (Cluster 3)
        'West Bengal', 'Assam', 'Mizoram', 'Meghalaya', 'Tripura',
        # Naxal-affected regions (Cluster 4)
        'Chhattisgarh', 'Jharkhand', 'Odisha', 'Andhra Pradesh', 'Telangana', 'Bihar', 'Maharashtra', 'Madhya Pradesh',
        # Myanmar border regions (Cluster 5)
```

```
    'Arunachal Pradesh', 'Nagaland', 'Manipur', 'Mizoram'
],
'Latitude': [
    # Pakistan border regions
    33.7782, 31.1471, 22.2587, 27.0238,
    # China border regions
    34.1526, 31.1048, 27.3389, 28.2180,
    # Bangladesh border regions
    22.9868, 26.2006, 23.1645, 25.4670, 23.9408,
    # Naxal-affected regions
    21.2787, 23.6102, 20.9517, 15.9129, 17.8495, 25.0961, 19.7515, 22.9734,
    # Myanmar border regions
    28.2180, 26.1584, 24.6637, 23.1645
],
'Longitude': [
    # Pakistan border regions
    76.5762, 75.3412, 71.1924, 74.2179,
    # China border regions
    77.5770, 77.1734, 88.6065, 94.7278,
    # Bangladesh border regions
    87.8550, 92.9376, 92.9376, 91.3662, 91.9882,
    # Naxal-affected regions
    81.8661, 85.2799, 85.0985, 79.7400, 79.1000, 85.3131, 75.7139, 78.6569,
    # Myanmar border regions
    94.7278, 94.5624, 93.9063, 92.9376
],
'Cluster': [
    # Pakistan border regions (Cluster 1)
    1, 1, 1, 1,
    # China border regions (Cluster 2)
```

```
        2, 2, 2, 2,
        # Bangladesh border regions (Cluster 3)
        3, 3, 3, 3, 3,
        # Naxal-affected regions (Cluster 4)
        4, 4, 4, 4, 4, 4, 4, 4,
        # Myanmar border regions (Cluster 5)
        5, 5, 5, 5
    ],
    'Threat_Type': [
        # Pakistan border regions
        'Pakistan Border', 'Pakistan Border', 'Pakistan Border', 'Pakistan Border',
        # China border regions
        'China Border', 'China Border', 'China Border', 'China Border',
        # Bangladesh border regions
        'Bangladesh Border', 'Bangladesh Border', 'Bangladesh Border', 'Bangladesh Border', 'Bangladesh Border',
        # Naxal-affected regions
        'Naxal Affected', 'Naxal Affected', 'Naxal Affected', 'Naxal Affected', 'Naxal Affected', 'Naxal Affected', 'Naxal
Affected', 'Naxal Affected',
        # Myanmar border regions
        'Myanmar Border', 'Myanmar Border', 'Myanmar Border', 'Myanmar Border'
    ]
}

# Create DataFrame
df = pd.DataFrame(regions_data)

# Create color mapping for clusters
cluster_colors = {
    1: 'red',     # Pakistan border
    2: 'blue',    # China border
    3: 'green',   # Bangladesh border
```

```
    4: 'purple',  # Naxal-affected
    5: 'orange'   # Myanmar border
}

# Create a map centered on India
map_india = folium.Map(location=[20.5937, 78.9629], zoom_start=5)

# Add markers for each region
for idx, row in df.iterrows():
    # Create popup text
    popup_text = f"Region: {row['Region']}<br>Threat Type: {row['Threat_Type']}"

    # Add marker
    folium.CircleMarker(
        location=[row['Latitude'], row['Longitude']],
        radius=8,
        popup=popup_text,
        color=cluster_colors[row['Cluster']],
        fill=True,
        fill_color=cluster_colors[row['Cluster']]
    ).add_to(map_india)

# Add legend
legend_html = '''
<div style="position: fixed;
    bottom: 50px; right: 50px; width: 150px; height: 90px;
    border:2px solid grey; z-index:9999; font-size:14px;
    background-color:white;
    ">
      <b>Cluster Legend</b> <br>
```

```
      Pakistan Intrusions   <i class="fa fa-circle fa-1x" style="color:red"></i><br>
      China Intrusions   <i class="fa fa-circle fa-1x" style="color:blue"></i><br>
      Bangladesh Intrusions   <i class="fa fa-circle fa-1x" style="color:green"></i><br>
      Naxal Affected   <i class="fa fa-circle fa-1x" style="color:purple"></i><br>
      Myanmar Intrusions   <i class="fa fa-circle fa-1x" style="color:orange"></i>
</div>
'''
map_india.get_root().html.add_child(folium.Element(legend_html))

# Save the map
map_india.save('india_security_clusters.html')

# Print cluster information
for cluster in range(5):
    print(f"\nCluster {cluster + 1} regions:")
    cluster_regions = df[df['Cluster'] == cluster + 1]['Region'].tolist()
    print(", ".join(cluster_regions))
```

```
!pip install pyspark
```

```
# Import necessary libraries
from pyspark.sql import SparkSession
from pyspark.sql.functions import col
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
import folium
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Initialize Spark session
spark = SparkSession.builder \
    .appName("India Attack Threat Analysis") \
    .getOrCreate()

# Load datasets
historic_attacks = spark.read.csv("India_Historic_Attacks.csv", header=True, inferSchema=True)
high_tension_areas = spark.read.csv("High_Tension_Places_India.csv", header=True, inferSchema=True)

# Create base map centered on India
india_map = folium.Map(location=[22.3511, 78.6677], zoom_start=5)

# Marker colors based on threat type
threat_colors = {
    "Foreign": "red",
    "Terrorist": "darkred",
    "Naxalite": "orange"
}

# Function to add markers to map
def add_markers(dataframe, map_object, is_historic=True):
    for row in dataframe.collect():
        threat_type = row['Attack_Type'] if is_historic else row['Threat_Type']
        color = threat_colors.get(threat_type, "blue")

        popup_text = f"""
        <b>{row['Event'] if is_historic else row['Location']}</b><br>
```

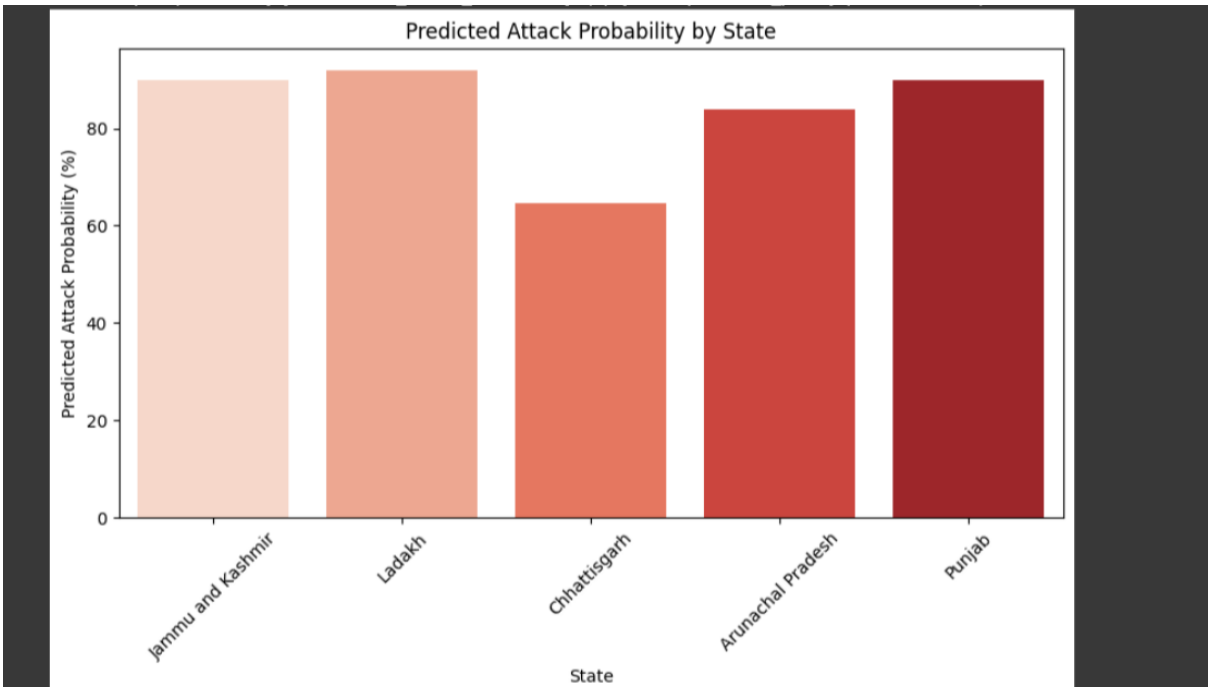
```
        Type: {threat_type}<br>
        State: {row['State']}<br>
        Location Type: {row['Location_Type']}<br>
        Threat Level: {row['Threat_Level']}<br>
        Notes: {row['Notes']}<br>
        News: {row['News_link']}
        """

        folium.CircleMarker(
            location=[row['Latitude'], row['Longitude']],
            radius=6,
            color=color,
            fill=True,
            fill_color=color,
            fill_opacity=0.7,
            popup=folium.Popup(popup_text, max_width=300)
        ).add_to(map_object)

# Add past attack markers
add_markers(historic_attacks, india_map, is_historic=True)

# Add current high-tension area markers
add_markers(high_tension_areas, india_map, is_historic=False)

# Save map to HTML file
india_map.save("India_Attack_Threat_Map.html")
print("Map saved as 'India_Attack_Threat_Map.html'")

# Prepare data for machine learning
data = [
```

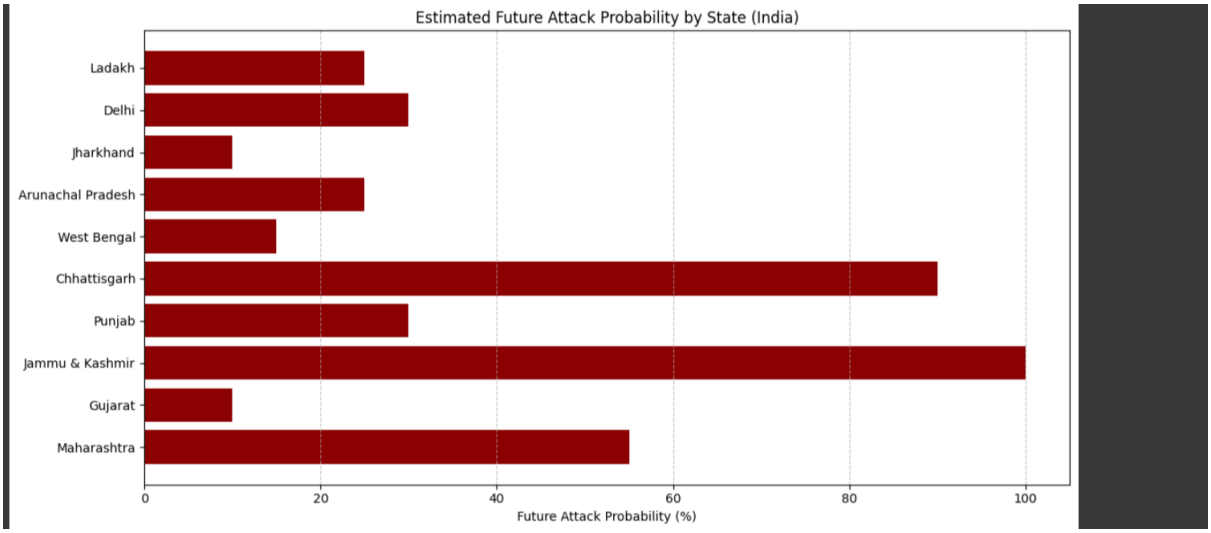
```
    ("Jammu and Kashmir", 1, 5, 1, 0.9, 0.1, 1, 1),
    ("Ladakh", 1, 3, 1, 0.7, 0.1, 1, 1),
    ("Chhattisgarh", 0, 8, 0, 0.2, 0.9, 0, 1),
    ("Arunachal Pradesh", 1, 2, 1, 0.6, 0.2, 1, 1),
    ("Punjab", 1, 4, 1, 0.8, 0.1, 0, 1),
    ("Assam", 1, 3, 0, 0.5, 0.3, 0, 0),
    ("Manipur", 0, 6, 0, 0.3, 0.4, 0, 0),
    ("Nagaland", 0, 5, 0, 0.3, 0.5, 0, 0),
    ("Maharashtra", 0, 2, 0, 0.1, 0.2, 0, 0),
    ("Rajasthan", 1, 1, 1, 0.4, 0.1, 1, 0)
]

columns = ["State", "Border_Proximity", "Past_Attacks", "Current_Troop_Buildup", "Terrorist_Infiltration_Risk",
"Naxalite_Risk", "Foreign_Threat", "Attack_Label"]
df = spark.createDataFrame(data, columns)

# Features and target variable
X = df.select("Border_Proximity", "Past_Attacks", "Current_Troop_Buildup", "Terrorist_Infiltration_Risk",
"Naxalite_Risk", "Foreign_Threat")
y = df.select("Attack_Label")

# Assemble features into a single vector
assembler = VectorAssembler(inputCols=X.columns, outputCol="features")
feature_df = assembler.transform(df)

# Train-Test Split
train_data, test_data = feature_df.randomSplit([0.7, 0.3], seed=42)

# Model (Random Forest Classifier for prediction)
rf = RandomForestClassifier(labelCol="Attack_Label", featuresCol="features", numTrees=100)
model = rf.fit(train_data)
```

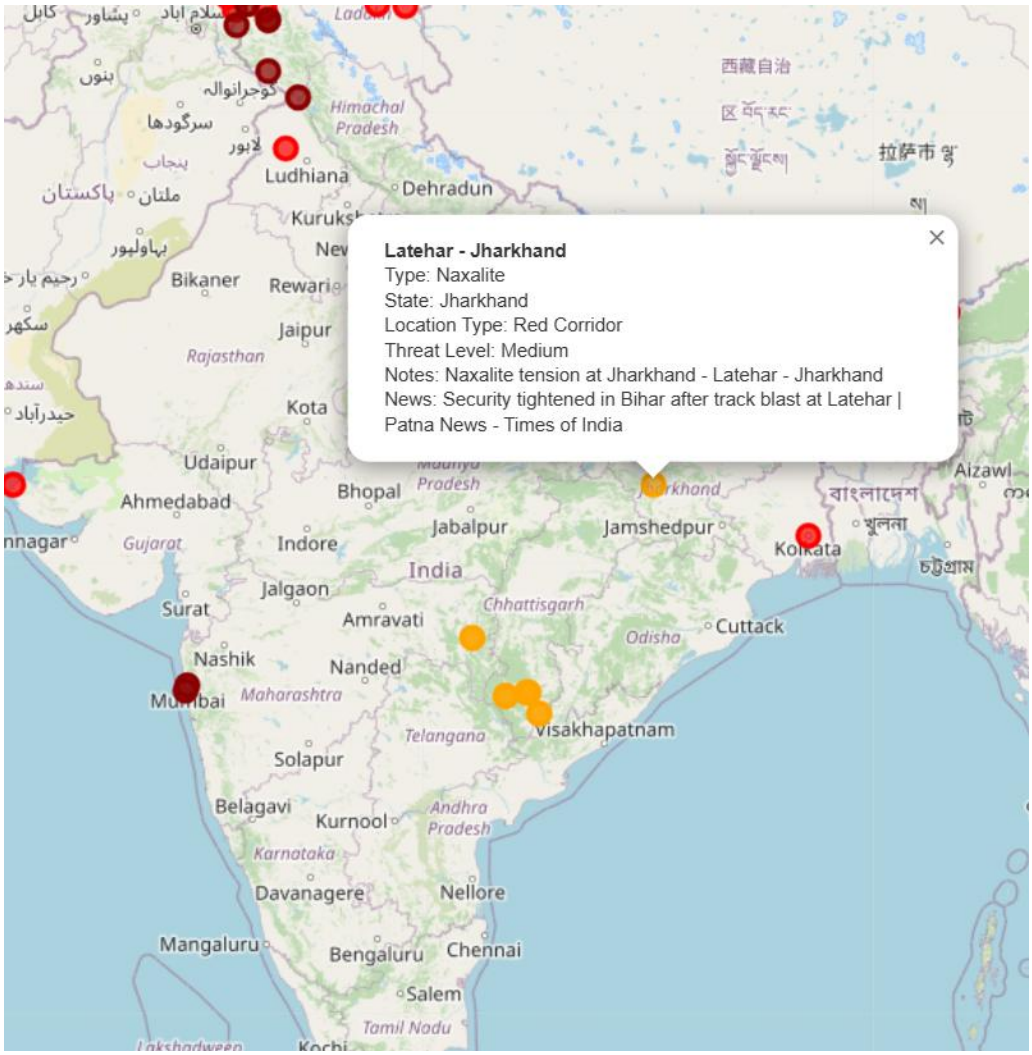
```
# Predictions on test set
predictions = model.transform(test_data)

# Evaluate the model
evaluator = MulticlassClassificationEvaluator(labelCol="Attack_Label", predictionCol="prediction",
metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print(f"Model Accuracy: {accuracy:.2f}")

# Predicting future attack probabilities for all states (including ones not in training set)
future_data = [
    ("Jammu and Kashmir", 1, 5, 1, 0.9, 0.1, 1),
    ("Ladakh", 1, 3, 1, 0.7, 0.1, 1),
    ("Chhattisgarh", 0, 8, 0, 0.2, 0.9, 0),
    ("Arunachal Pradesh", 1, 2, 1, 0.6, 0.2, 1),
    ("Punjab", 1, 4, 1, 0.8, 0.1, 0)
]

future_columns = ["State", "Border_Proximity", "Past_Attacks", "Current_Troop_Buildup",
"Terrorist_Infiltration_Risk", "Naxalite_Risk", "Foreign_Threat"]
future_df = spark.createDataFrame(future_data, future_columns)

# Assemble features for future predictions
future_feature_df = assembler.transform(future_df)

# Predict probabilities for future attacks
future_predictions = model.transform(future_feature_df)
future_predictions.select("State", "prediction", "probability").show()

# Visualization of predicted risk levels
```

```
predicted_probs = future_predictions.select("State", "probability").toPandas()
predicted_probs['Predicted_Attack_Probability (%)'] = predicted_probs['probability'].apply(lambda x: x[1] * 100)

plt.figure(figsize=(10, 5))
sns.barplot(x="State", y="Predicted_Attack_Probability (%)", data=predicted_probs, palette="Reds")
plt.xticks(rotation=45)
plt.title("Predicted Attack Probability by State")
plt.ylabel("Predicted Attack Probability (%)")
plt.show()

# Count historical attacks and high-tension areas per state
historic_count = historic_attacks.groupBy("State").count().toPandas().set_index('State').to_dict()['count']
tension_count = high_tension_areas.groupBy("State").count().toPandas().set_index('State').to_dict()['count']

# Combine counts into a single dataframe for visualization
states = set(historic_count.keys()).union(set(tension_count.keys()))
state_data = []

for state in states:
    past_attacks = historic_count.get(state, 0)
    current_tensions = tension_count.get(state, 0)
    future_risk = (past_attacks * 0.6) + (current_tensions * 0.4)  # Weighted scoring (adjustable)
    state_data.append([state, past_attacks, current_tensions, future_risk])

risk_df = pd.DataFrame(state_data, columns=['State', 'Past_Attacks', 'Current_Tensions', 'Future_Risk'])

# Normalize future risk to % (just for better visualization)
max_risk = risk_df['Future_Risk'].max()
risk_df['Future_Risk_Percentage'] = (risk_df['Future_Risk'] / max_risk) * 100
```

```
# Visualization of future attack chances
plt.figure(figsize=(12, 6))
plt.barh(risk_df['State'], risk_df['Future_Risk_Percentage'], color='darkred')
plt.xlabel('Future Attack Probability (%)')
plt.title('Estimated Future Attack Probability by State (India)')
plt.grid(axis='x', linestyle='--', alpha=0.7)

# Save graph to file
plt.tight_layout()
plt.savefig("Future_Attack_Probability.png")
plt.show()

print("Future attack probability graph saved as 'Future_Attack_Probability.png'")
```




Estimated Future Attack Probability by State (India)
Ladakh
Delhi
Jharkhand
Arunachal Pradesh
West Bengal
Chhattisgarh
Punjab
Jammu & Kashmir
Gujarat
Maharashtra
0
20
40
60
80
100
Future Attack Probability (%)


Latehar - Jharkhand
Type: Naxalite
State: Jharkhand
Location Type: Red Corridor
Threat Level: Medium
Notes: Naxalite tension at Jharkhand - Latehar - Jharkhand
News: Security tightened in Bihar after track blast at Latehar | Patna News - Times of India
Ludhiana
Dehradun
Himachal Pradesh
Bikaner
Jaipur
Rajasthan
Kota
Udaipur
Ahmedabad
Bhopal
Indore
Jabalpur
Jamshedpur
Kolkata
India
Surat
Jalgaon
Amravati
Chhattisgarh
Odisha
Cuttack
Nashik
Nanded
Maharashtra
Visakhapatnam
Telangana
Solapur
Belagavi
Kurnool
Andhra Pradesh
Karnataka
Davanagere
Nellore
Mangaluru
Bengaluru
Chennai
Salem
Tamil Nadu

Conclusion:

Through the application of K-means clustering and predictive modeling, this study identifies states most vulnerable to future attacks based on historical data. The findings offer valuable insights for policymakers and law enforcement agencies to enhance counter-terrorism strategies, allocate resources effectively, and strengthen national security measures.

ASSIGNMENT

CASE STUDY

(Explain One Real-World Application of Using Apache Spark for Machine Learning)

Theory: Fraud detection is a critical application in the financial sector, where organizations aim to identify and prevent fraudulent activities in real-time. With the increasing volume of transactions, traditional methods of fraud detection are often insufficient. Apache Spark provides a powerful framework for processing large datasets efficiently, enabling the implementation of machine learning algorithms that can analyze transaction patterns and detect anomalies indicative of fraud. By leveraging Spark's distributed computing capabilities, organizations can build scalable and robust fraud detection systems that operate in real-time.

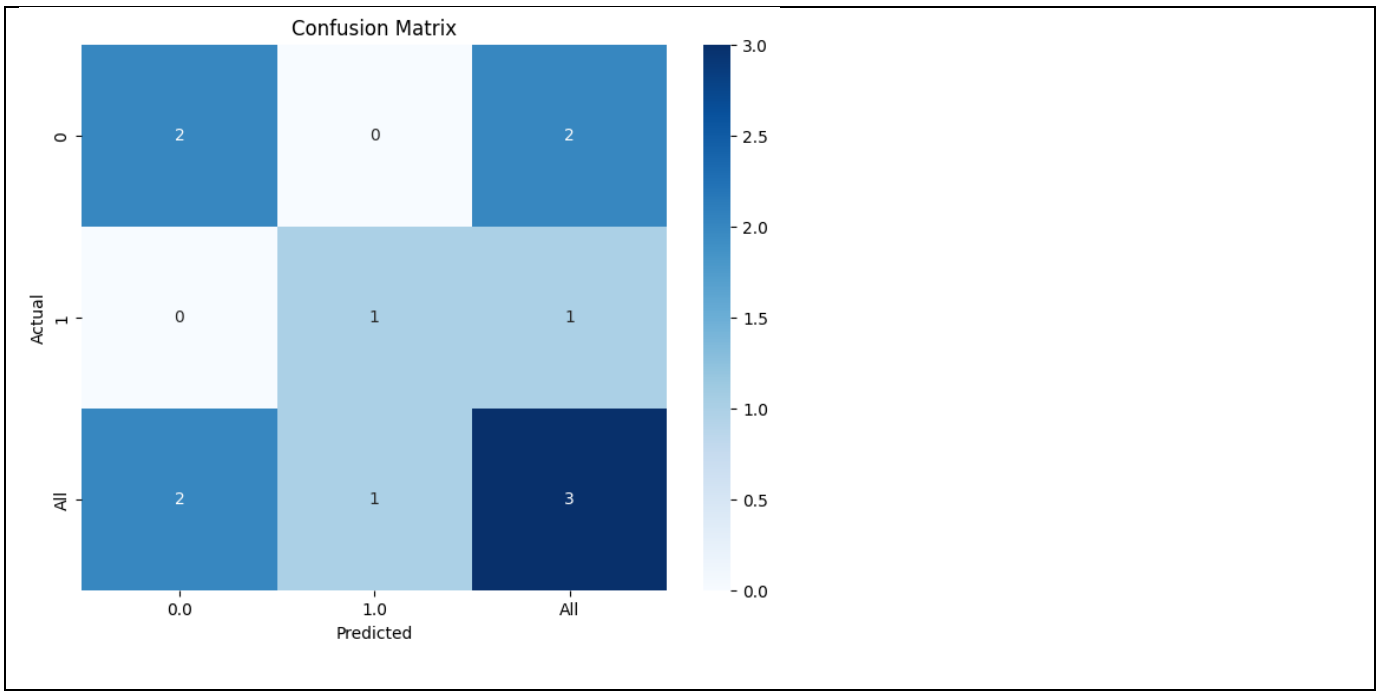
Objectives:

- To understand the application of machine learning in fraud detection using Apache Spark.
- To implement a fraud detection model using a Random Forest classifier.
- To preprocess transaction data, including feature engineering and encoding categorical variables.
- To evaluate the model's performance using metrics such as Area Under the ROC Curve (AUC) and visualize the results with a confusion matrix.

Code and Output:

```
# Import necessary libraries
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StringIndexer, OneHotEncoder
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.sql.functions import when
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns # Import seaborn library

# Initialize SparkSession
spark = SparkSession.builder \
    .appName("Fraud Detection") \
    .getOrCreate()

# Load synthetic transaction data
data = spark.read.csv("transactions.csv", header=True, inferSchema=True)

# Preprocess data
data = data.withColumn("is_fraud_numeric", when(data["is_fraud"] == True, 1).otherwise(0))
merchant_indexer = StringIndexer(inputCol="merchant", outputCol="merchant_index")
category_indexer = StringIndexer(inputCol="category", outputCol="category_index")
merchant_encoder = OneHotEncoder(inputCol="merchant_index", outputCol="merchant_encoded")
category_encoder = OneHotEncoder(inputCol="category_index", outputCol="category_encoded")
```

```
assembler = VectorAssembler(inputCols=['amount', 'merchant_encoded', 'category_encoded'],
outputCol='features')

# Create a pipeline
pipeline = Pipeline(stages=[merchant_indexer, category_indexer, merchant_encoder, category_encoder,
assembler])

# Fit and transform data
pipeline_model = pipeline.fit(data)
data = pipeline_model.transform(data)

# Split data into training and test sets
train_data, test_data = data.randomSplit([0.8, 0.2], seed=42)

# Train Random Forest classifier
rf = RandomForestClassifier(labelCol='is_fraud_numeric', featuresCol='features')
model = rf.fit(train_data)

# Evaluate model performance
predictions = model.transform(test_data)
evaluator = BinaryClassificationEvaluator(labelCol='is_fraud_numeric')
auc = evaluator.evaluate(predictions)
print(f"Area Under ROC: {auc}")

# Convert predictions to Pandas DataFrame for visualization
predictions_pd = predictions.select("is_fraud_numeric", "prediction").toPandas()

# Create a confusion matrix
confusion_matrix = pd.crosstab(predictions_pd['is_fraud_numeric'], predictions_pd['prediction'],
                               rownames=['Actual'], colnames=['Predicted'], margins=True)

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
plt.title('Confusion Matrix')
sns.heatmap(confusion_matrix, annot=True, fmt='d', cmap='Blues') # Now sns is defined and can be
used
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

# Stop the SparkSession
spark.stop()
```



Conclusion:

In this case study, we demonstrated the application of Apache Spark for fraud detection using machine learning. By implementing a Random Forest classifier, we were able to preprocess transaction data, train a model, and evaluate its performance effectively. The visualization of the confusion matrix provided insights into the model's accuracy in distinguishing between fraudulent and non-fraudulent transactions. This approach highlights the power of Apache Spark in handling large datasets and performing real-time analytics, making it an invaluable tool for organizations aiming to enhance their fraud detection capabilities.