BIG DATA ANALYSIS

Practical Journal

Submitted By

Shaba Khan

Class: MSC. CS (Sem III)

Roll No. : 31031523039

Department of computer Science

Somaiya Vidyavihar University

S K Somaiya College

NAME: Shaba Khan
ROLL NO. : 31031523039

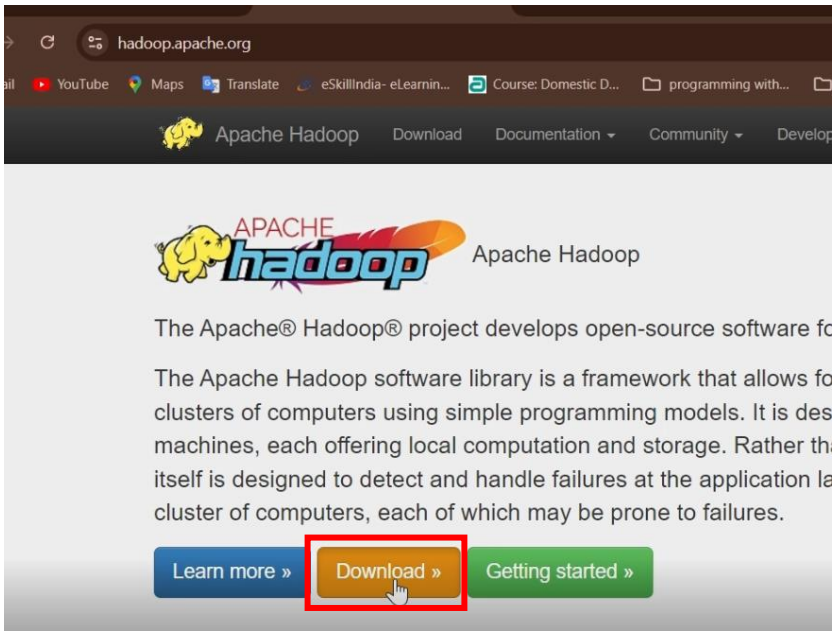
Practical 01

AIM: To install and configure the Apache Hadoop framework.

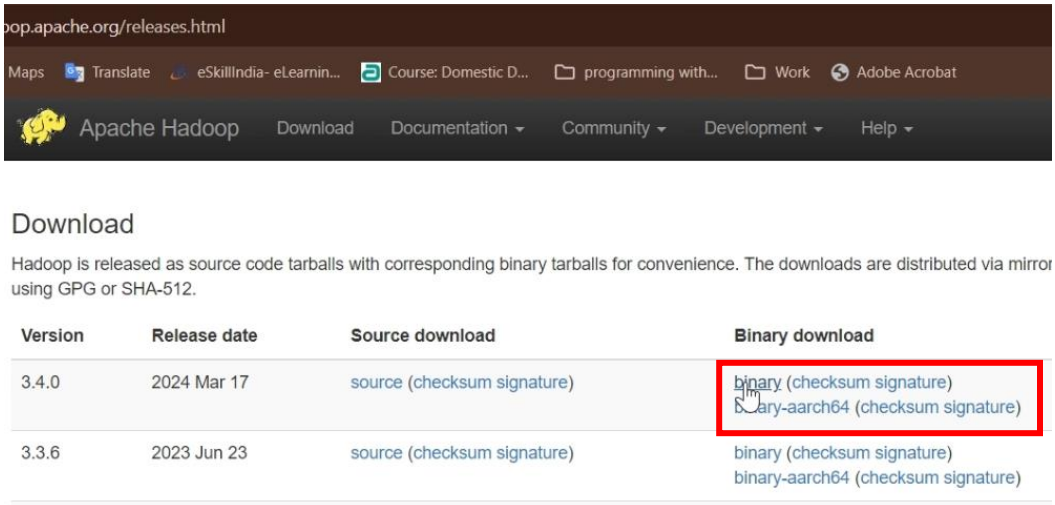
STEPS:

Step 01: Download Hadoop Binaries

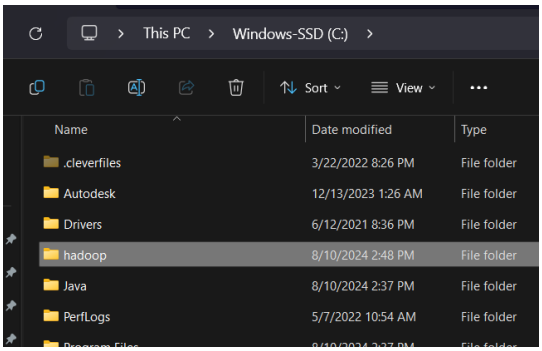
(i) Visit the [Apache Hadoop official website](https://hadoop.apache.org/releases.html).



(ii) Download the latest stable version of Hadoop (e.g., Hadoop 3.x).



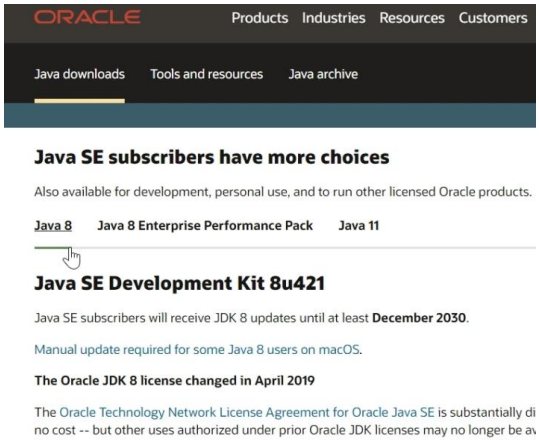
(iii) Extract the downloaded Hadoop binaries to a directory (e.g., `C:\hadoop`).



Step 02: Install Java Development Kit (JDK) 8

(i) Download JDK 8:

→ Visit the [Oracle JDK 8 Downloads page](https://www.oracle.com/java/technologies/javase/javase8-archive-downloads.html).

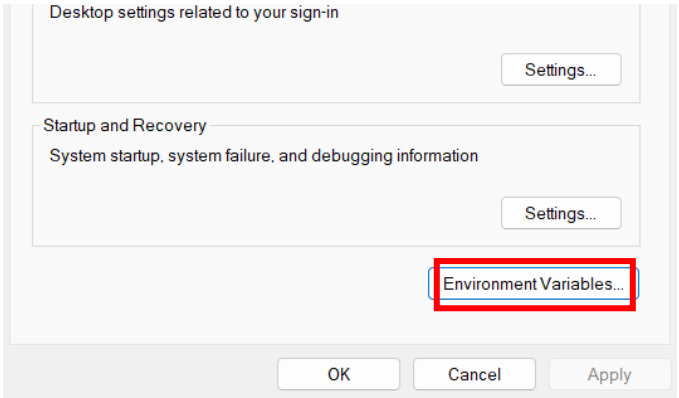


(ii) Install JDK 8:

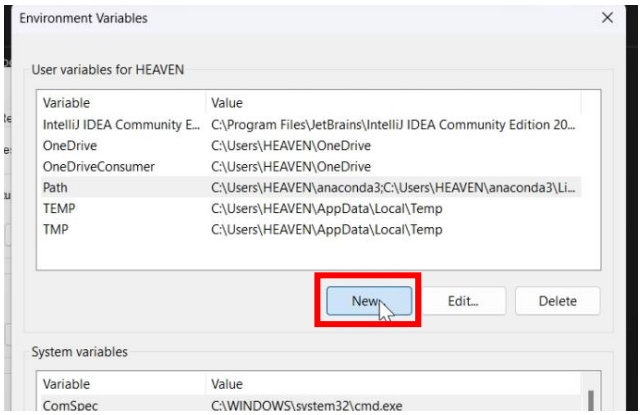
→ Run the downloaded installer. Follow the installation wizard to install JDK 8. Note the installation directory (e.g., `C:\ \Java\jdk1.8`).

(iii) Set the `JAVA_HOME` Environment Variable:

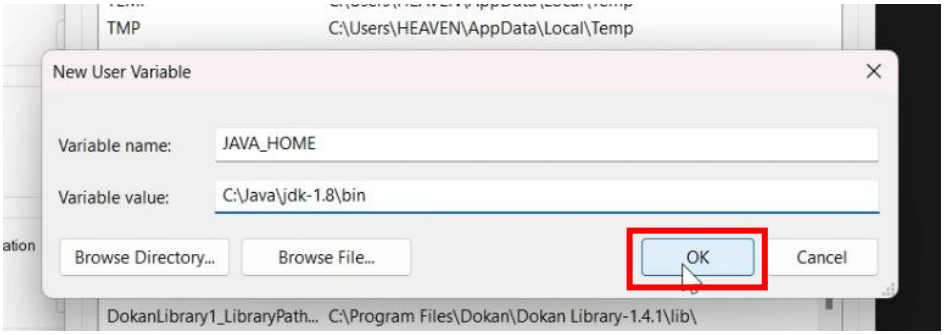
→ Search for “Edit the system Environmental variables”. Click on the “Environment Variables..”



→ Under "System variables", click "New" and set:



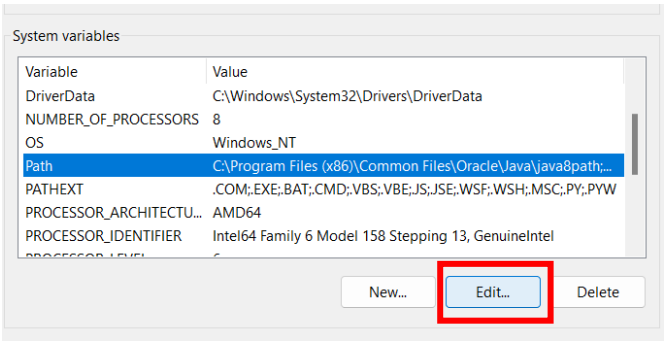
Variable name: `JAVA_HOME`
Variable value: `C: \Java\jdk1.8\bin`



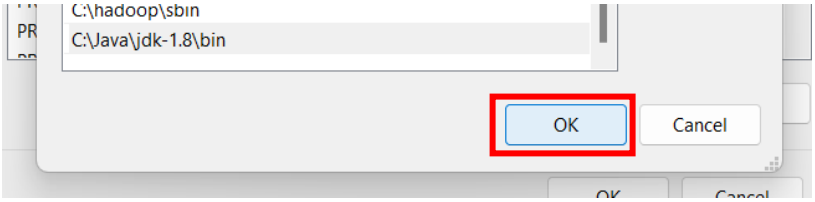
→ Click OK.

(iv) Update the `Path` Variable:

→ In the "Environment Variables" window, find the `Path` variable under "System variables" and select it, then click "Edit".



→ Add `%JAVA_HOME%\bin` to the `Path` variable. Ensure that the new entry is separated from existing entries by a semicolon.

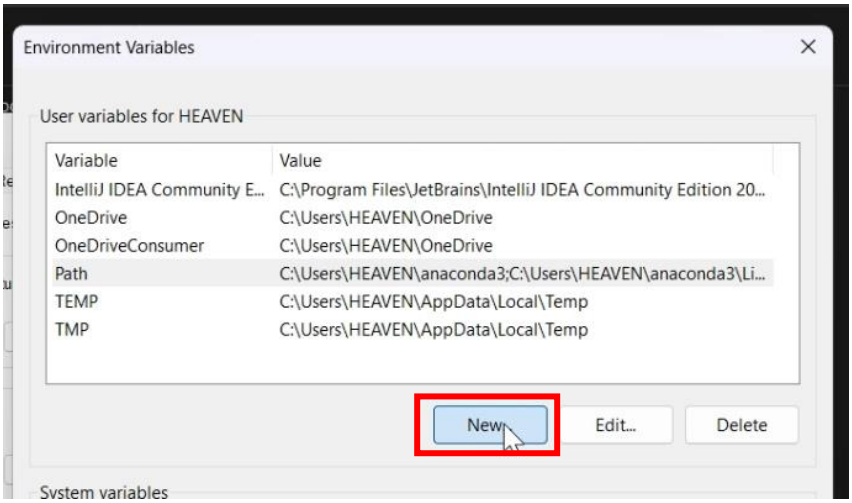


→ Click OK to save changes.

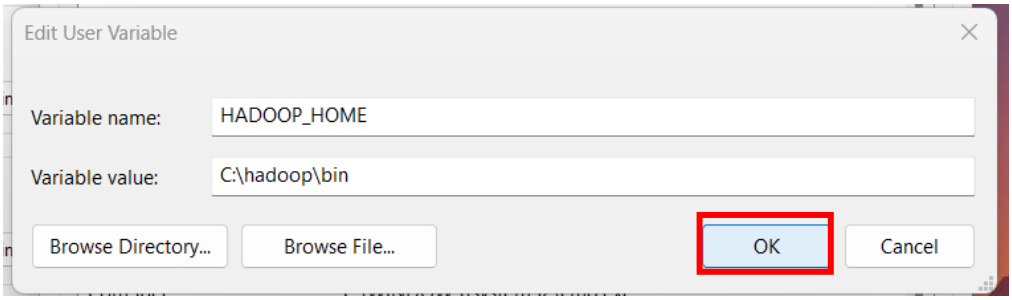
Step 03: Configure Hadoop Environment Variables

(i) Set Hadoop environment variables:

→ Under "System variables", click "New" and set:



Variable name: `HADOOP_HOME`
Variable value: `C:\hadoop`

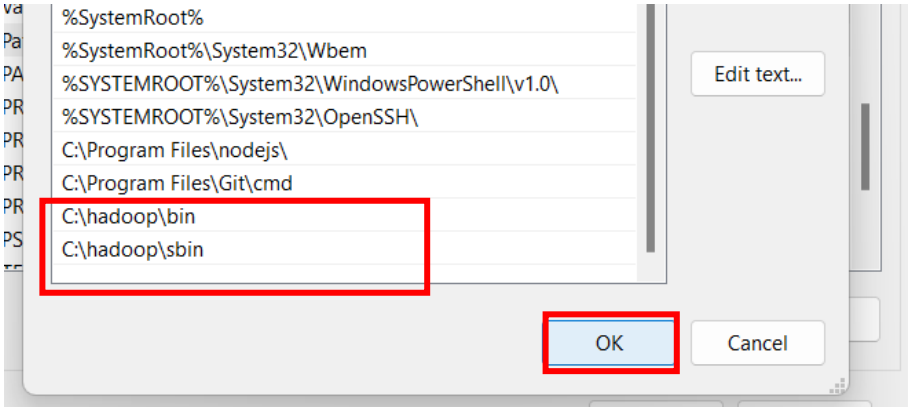


→ Click OK.

(ii) Edit the `Path` variable under "System variables":

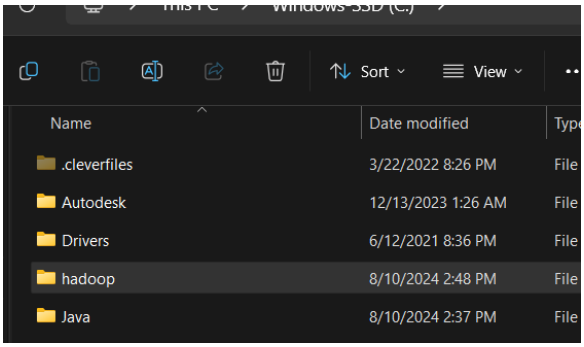
→ Add ` C:\hadoop\bin` to the `Path`.

→ Add 'C:\hadoop\sbin' to the 'Path'

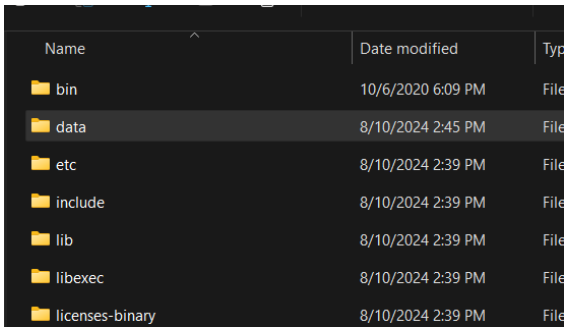


Step 04: Create Hadoop Data Folders

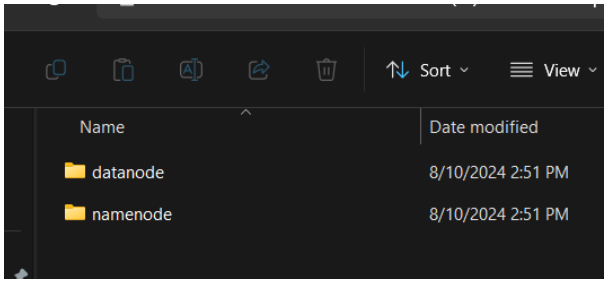
(i) Navigate to C:\hadoop in File Explorer.



(ii) Create a new folder named 'data'.



(iii) Inside the 'data' folder, create two new folders: 'namenode' and 'datanode'.



Step 05: Configure Hadoop

(i) Go to the Hadoop directory (` C:\hadoop\etc\hadoop `).

(ii) Edit the following configuration files:

→ ***core-site.xml:***

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

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

→ ***hdfs-site.xml:***

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

<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>
```

→ ***mapred-site.xml:***

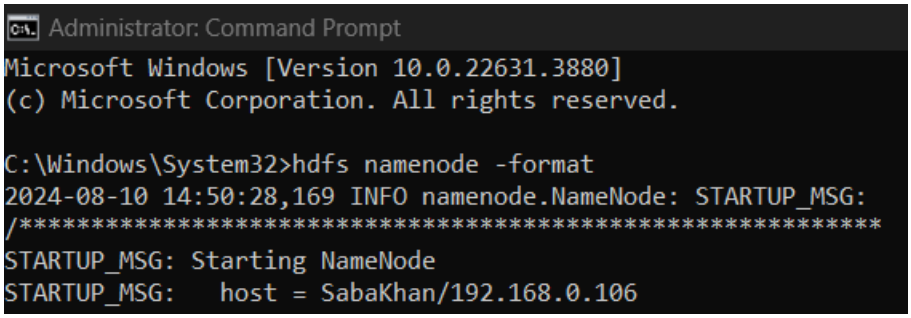
```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="configuration.xsl"?>

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

→ ***yarn-site.xml:***

```
<?xml version="1.0"?>

<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>
```

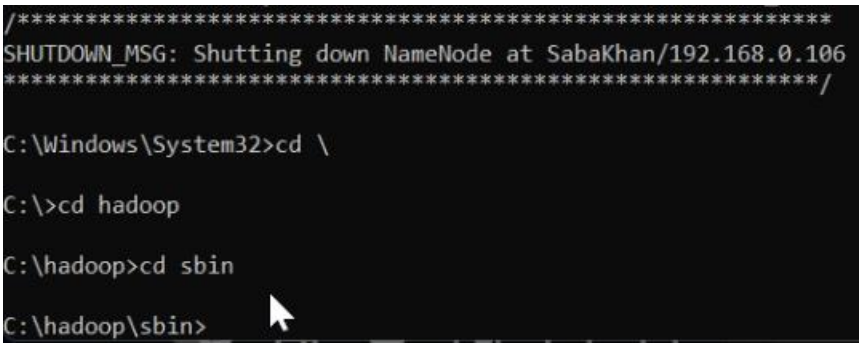
Step 06: Format the Namenode

(i) Open a command prompt and Run as administrator.
(ii) Run the following command to format the namenode:
- hdfs namenode -format



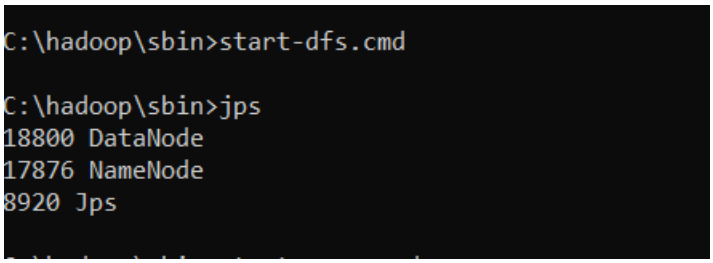
Step 07: Start Hadoop Services

(i) Open the command prompt and navigate to the Hadoop directory (` C:\hadoop\sbin `).

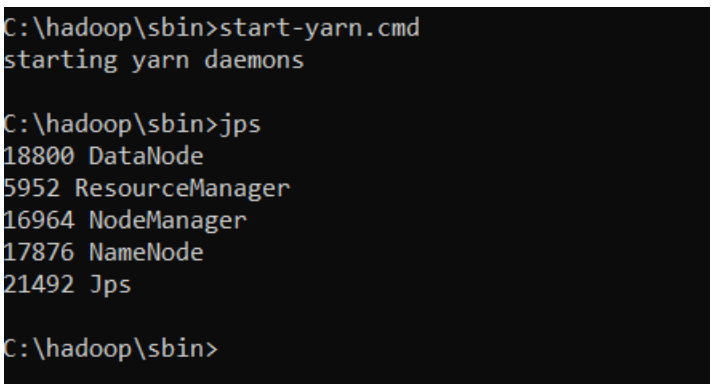


(ii) Start Hadoop services using the following commands:

→ start-dfs.cmd



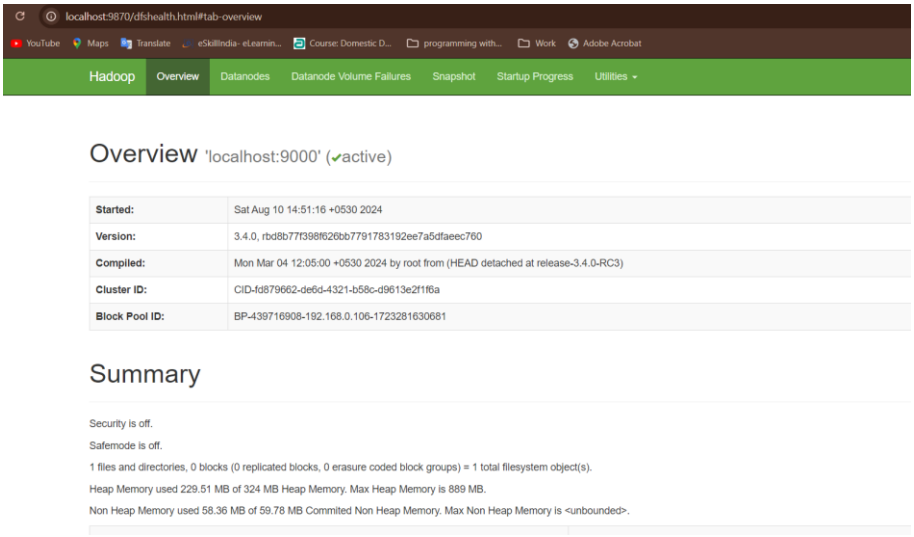
→ start-yarn.cmd



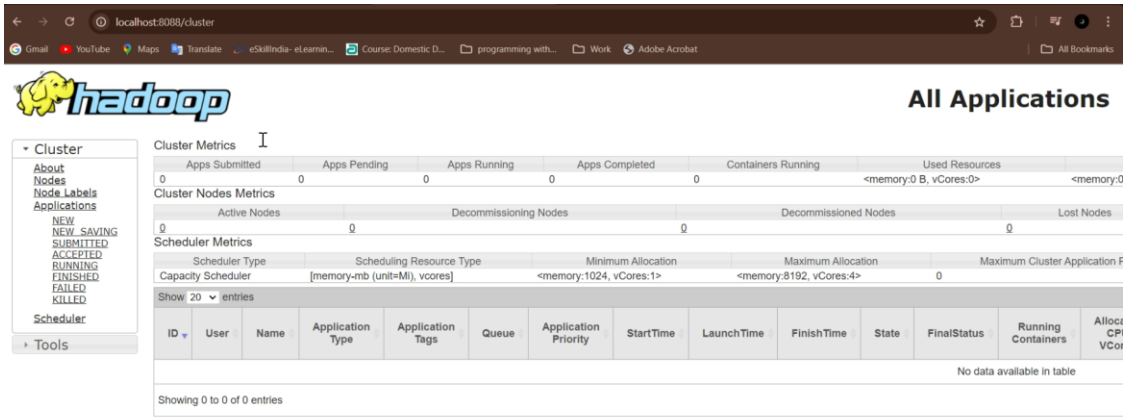
Step 08: Verify Hadoop Installation

(i) Open a web browser and verify the following:

→ HDFS: `http://localhost:9870`



→ YARN: `http://localhost:8088`



Step 09: Stop Hadoop Services

(i) To stop Hadoop services, run the following commands:

→ stop-yarn.cmd

```
C:\hadoop\sbin>stop-yarn.cmd
stopping yarn daemons
SUCCESS: Sent termination signal to the process with PID 22240.
SUCCESS: Sent termination signal to the process with PID 14084.

INFO: No tasks running with the specified criteria.

C:\hadoop\sbin>
```

→ stop-dfs.cmd

```
C:\hadoop\sbin>stop-dfs.cmd
SUCCESS: Sent termination signal to the process with PID 14088.
SUCCESS: Sent termination signal to the process with PID 16568.

C:\hadoop\sbin>
```

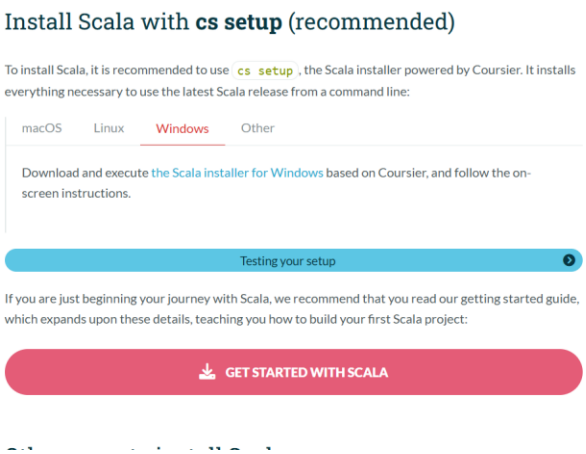
NAME: Shaba Khan
ROLL NO. : 31031523039

Practical 02

AIM: SparkSQL

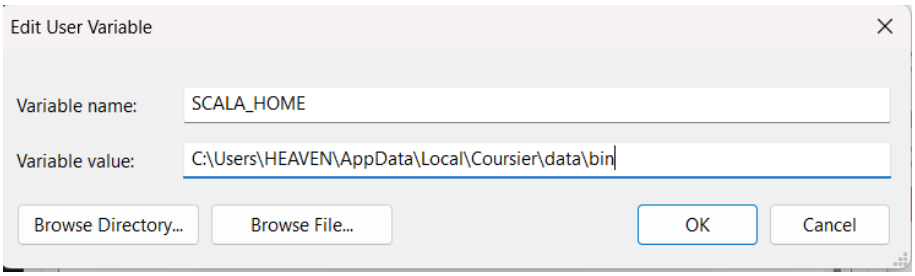
STEPS:

Step 01: Download scala for windows (https://www.scala-lang.org/download/)

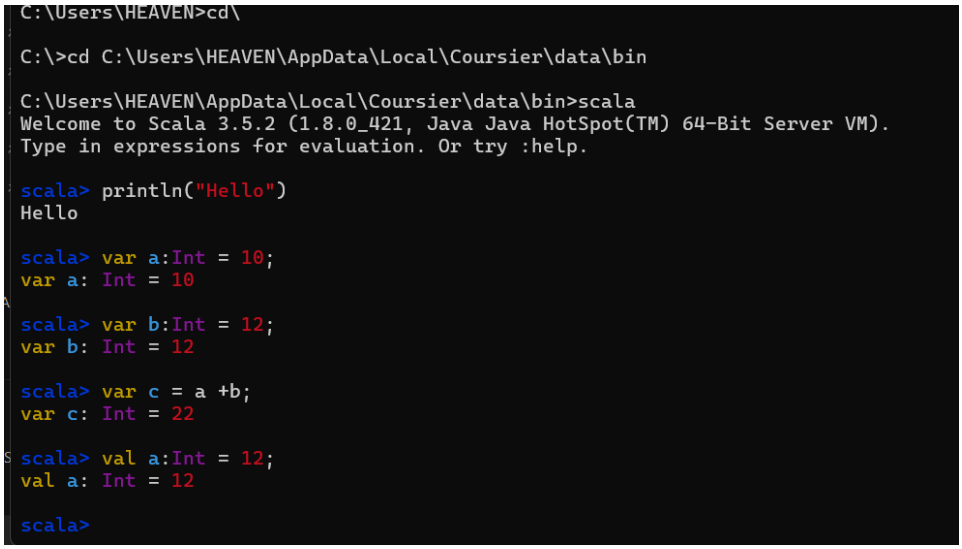


Step 02: Extract the file and the set up the environment variables:

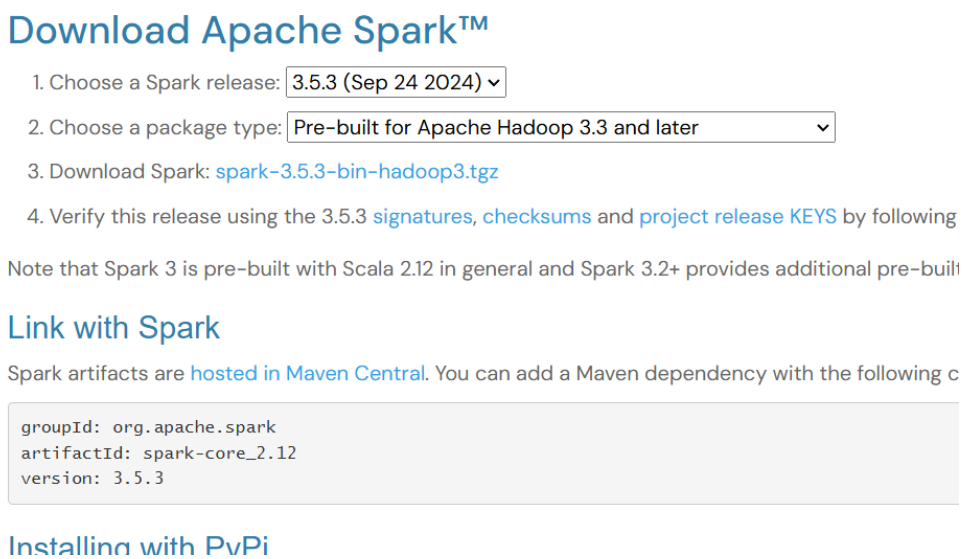
- SCALA_HOME (C:\Users\admin\AppData\Local\Coursier\data\bin)



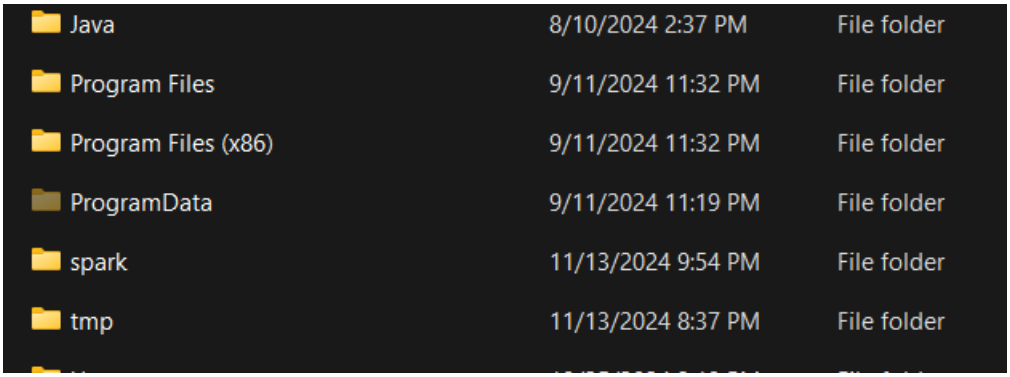
Step 03: Try working with commands



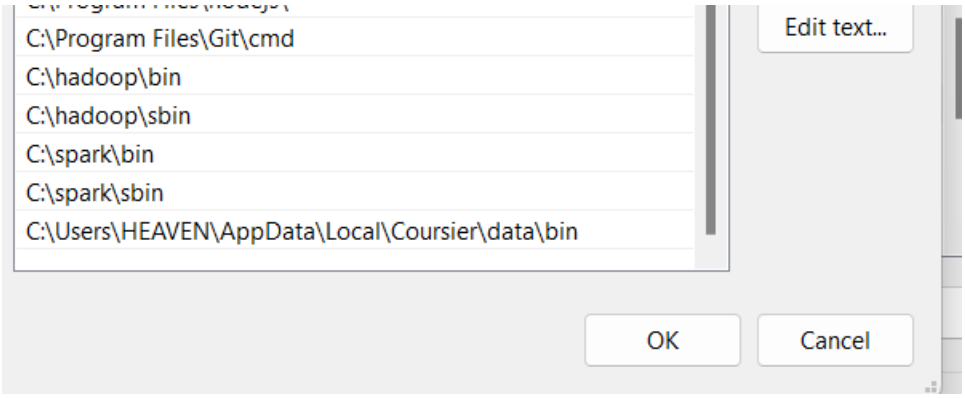
Step 03: Install Apache sparks (https://spark.apache.org/downloads.html)



Step 04: Extract the Spark file to the “C” Directory .



Step 05: Set the environment variable for spark as well.



Step 06: Check the installation with 'spark-shell'

```
C:\Users\HEAVEN>spark-shell
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /___/ .__/\_,_/_/ /_/\_\   version 3.5.3
      /_/

Using Scala version 2.13.8 (Java HotSpot(TM) 64-Bit Server VM, Java 1.8.0_421)
Type in expressions to have them evaluated.
Type :help for more information.
Spark context Web UI available at http://SabaKhan.lan:4040
Spark context available as 'sc' (master = local[*], app id = local-1731518604689).
Spark session available as 'spark'.

scala>
```

Step 07: Lets try to show a sample data from this path

- C:\spark\spark-3.4.3-bin-hadoop3\examples\src\main\resources\people.json

```
scala> val x = spark.read.json("C:/spark/examples/src/main/resources/people.json");
val x: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> x.show()
+----+-------+
| age|   name|
+----+-------+
|NULL|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

```
scala> x.printSchema()
root
 |-- age: long (nullable = true)
 |-- name: string (nullable = true)
```

```
scala> x.select($"name",$"age").show()
+-------+----+
|   name| age|
+-------+----+
|Michael|NULL|
|   Andy|  30|
| Justin|  19|
+-------+----+
```

```
scala> x.filter($"age">20).show()
+---+----+
|age|name|
+---+----+
| 30|Andy|
+---+----+
```

Step 08: Reading CSV/Excel File

```
scala> val y = spark.read.csv("C:/spark/examples/src/main/resources/people.csv").sho
w()
+-----------------+
|              _c0|
+-----------------+
|      name;age;job|
|Jorge;30;Developer|
|  Bob;32;Developer|
+-----------------+

val y: Unit = ()
```

Step 09: Creating an SQL Tempory View

```
scala> x.createOrReplaceTempView("people")

scala> val sqlDF = spark.sql("Select * from people")
val sqlDF: org.apache.spark.sql.DataFrame = [age: bigint, name: string]

scala> sqlDF.show()
+----+-------+
| age|   name|
+----+-------+
|NULL|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

Creating Datasets

Datasets are similar to RDDs, however, instead of using Java serialization or Kryo they use a specialized Encoder to serialize the objects for processing or transmitting over the network. While both encoders and standard serialization are responsible for turning an object into bytes,

encoders are code generated dynamically and use a format that allows Spark to perform many operations like filtering, sorting and hashing without deserializing the bytes back into an object.

```
case class Person(name: String, age: Long)

// Encoders are created for case classes
val caseClassDS = Seq(Person("Andy", 32)).toDS()
caseClassDS.show()
// +----+---+
// |name|age|
// +----+---+
// |Andy| 32|
// +----+---+

// Encoders for most common types are automatically provided by importing spark.implicits._
val primitiveDS = Seq(1, 2, 3).toDS()
primitiveDS.map(_ + 1).collect() // Returns: Array(2, 3, 4)

// DataFrames can be converted to a Dataset by providing a class. Mapping will be done by name
val path = "examples/src/main/resources/people.json"
val peopleDS = spark.read.json(path).as[Person]
peopleDS.show()
// +----+-------+
// | age| name|
// +----+-------+
// |null|Michael|
// | 30| Andy|
// | 19| Justin|
// +----+-------+
```

```
scala> case class person(name:String, age:Long)
class person

scala> val caseClaseeDS = Seq(Person("Olivia",25)).toDS()
                              ^
       error: not found: value Person

scala> val caseClaseeDS = Seq(person("Olivia",25)).toDS()
val caseClaseeDS: org.apache.spark.sql.Dataset[person] = [name: string, age: bigint]

scala> caseClassDS.show()
       ^
       error: not found: value caseClassDS

scala> caseClaseeDS.show()
+------+---+
|  name|age|
+------+---+
|Olivia| 25|
+------+---+
```

```
scala> val primitiveDS = Seq(1,2,3).toDS()
val primitiveDS: org.apache.spark.sql.Dataset[Int] = [value: int]

scala> primitiveDS.map(_ + 1).collect()
val res11: Array[Int] = Array(2, 3, 4)
```

```
scala> val peopleDS = spark.read.json("C:/spark/examples/src/main/resources/people.json").
as[person]
val peopleDS: org.apache.spark.sql.Dataset[person] = [age: bigint, name: string]

scala> peopleDS.show()
+----+-------+
| age|   name|
+----+-------+
|NULL|Michael|
|  30|   Andy|
|  19| Justin|
+----+-------+
```

Inferring the Schema Using Reflection
The Scala interface for Spark SQL supports automatically converting an RDD containing case classes to a DataFrame. The case class defines the schema of the table. The names of the arguments to the case class are read using reflection and become the names of the columns.

Case classes can also be nested or contain complex types such as Seqs or Arrays. This RDD can be implicitly converted to a DataFrame and then be registered as a table. Tables can be used in subsequent SQL statements.

// For implicit conversions from RDDs to DataFrames
import spark.implicits._

```
// Create an RDD of Person objects from a text file, convert it to a Dataframe
val peopleDF = spark.sparkContext
.textFile("examples/src/main/resources/people.txt")
.map(_.split(","))
.map(attributes => Person(attributes(0), attributes(1).trim.toInt))
.toDF()
// Register the DataFrame as a temporary view
peopleDF.createOrReplaceTempView("people")

// SQL statements can be run by using the sql methods provided by Spark
val teenagersDF = spark.sql("SELECT name, age FROM people WHERE age BETWEEN 13
AND 19")

// The columns of a row in the result can be accessed by field index
teenagersDF.map(teenager => "Name: " + teenager(0)).show()
// +------------+
// | value|
// +------------+
// |Name: Justin|
// +------------+

// or by field name
teenagersDF.map(teenager => "Name: " + teenager.getAs[String]("name")).show()
// +------------+
// | value|
// +------------+
// |Name: Justin|
// +------------+

// No pre-defined encoders for Dataset[Map[K,V]], define explicitly
implicit val mapEncoder = org.apache.spark.sql.Encoders.kryo[Map[String, Any]]
// Primitive types and case classes can be also defined as
// implicit val stringIntMapEncoder: Encoder[Map[String, Any]] = ExpressionEncoder()

// row.getValuesMap[T] retrieves multiple columns at once into a Map[String, T]
teenagersDF.map(teenager => teenager.getValuesMap[Any](List("name", "age"))).collect()
// Array(Map("name" -> "Justin", "age" -> 19))
```

```
scala> case class Person(name: String, age: Long)
defined class Person

scala> val peopleDF = spark.sparkContext.textFile("C:/spark/spark-3.4.3-bin-hadoop3/examples/src/main/resources/people.t
xt").map(_.split(",")).map(attributes => Person(attributes(0), attributes(1).trim.toInt)).toDF()
peopleDF: org.apache.spark.sql.DataFrame = [name: string, age: bigint]

scala> peopleDF.createOrReplaceTempView("people")

scala> val teenagersDF = spark.sql("SELECT name, age FROM people WHERE age BETWEEN 13 AND 19")
teenagersDF: org.apache.spark.sql.DataFrame = [name: string, age: bigint]

scala> teenagersDF.map(teenager => "Name: " + teenager(0)).show()
+------------+
|       value|
+------------+
|Name: Justin|
+------------+
```

```
scala> teenagersDF.map(teenager => "Name: " + teenager.getAs[String]("name")).show()
+------------+
|       value|
+------------+
|Name: Justin|
+------------+


scala> implicit val mapEncoder = org.apache.spark.sql.Encoders.kryo[Map[String, Any]
     |
     | ]
mapEncoder: org.apache.spark.sql.Encoder[Map[String,Any]] = class[value[0]: binary]

scala> teenagersDF.map(teenager => teenager.getValuesMap[Any](List("name", "age"))).collect()
res3: Array[Map[String,Any]] = Array(Map(name -> Justin, age -> 19))
```

Programmatically Specifying the Schema
When case classes cannot be defined ahead of time (for example, the structure of records is encoded in a string, or a text dataset will be parsed and fields will be projected differently for different users), a DataFrame can be created programmatically with three steps.

Create an RDD of Rows from the original RDD;
Create the schema represented by a StructType matching the structure of Rows in the RDD created in Step 1.
Apply the schema to the RDD of Rows via createDataFrame method provided by SparkSession.

```
import org.apache.spark.sql.Row
import org.apache.spark.sql.types._

// Create an RDD
val peopleRDD = spark.sparkContext.textFile("examples/src/main/resources/people.txt")

// The schema is encoded in a string
val schemaString = "name age"
```

```
// Generate the schema based on the string of schema
val fields = schemaString.split(" ")
.map(fieldName => StructField(fieldName, StringType, nullable = true))

val schema = StructType(fields)

// Convert records of the RDD (people) to Rows
val rowRDD = peopleRDD
.map(_.split(","))
.map(attributes => Row(attributes(0), attributes(1).trim))

// Apply the schema to the RDD
val peopleDF = spark.createDataFrame(rowRDD, schema)

// Creates a temporary view using the DataFrame
peopleDF.createOrReplaceTempView("people")

// SQL can be run over a temporary view created using DataFrames
val results = spark.sql("SELECT name FROM people")

// The results of SQL queries are DataFrames and support all the normal RDD operations
// The columns of a row in the result can be accessed by field index or by field name
results.map(attributes => "Name: " + attributes(0)).show()
// +-------------+
// | value|
// +-------------+
// |Name: Michael|
// | Name: Andy|
// | Name: Justin|
// +-------------+

import org.apache.spark.sql.Row
Import org.apache.spark.sql.types._
```

```
scala> import org.apache.spark.sql.Row
import org.apache.spark.sql.Row

scala> Import org.apache.spark.sql.types._
               ^
       error: ';' expected but '.' found.

scala> import org.apache.spark.sql.types._
import org.apache.spark.sql.types._
```

val peopleRDD =
spark.sparkContext.textFile(“C:/spark/hadoop/examples/src/main/resources/people.txt”)

```
scala> val peopleRDD = spark.sparkContext.textFile("C:/spark/hadoop/examples/src/main/res
o rces/people.txt")
val peopleRDD: org.apache.spark.rdd.RDD[String] = C:/spark/hadoop/examples/src/main/resou
rces/people.txt MapPartitionsRDD[48] at textFile at <console>:1
```

val fields = schemaString.split(“ “).map(fieldName => StructField(fieldName, StringType, nullable = true))

```
scala> val fields = schemaString.split(" ").map(fieldName => StructField(fieldName, StringType, nullab
le = true))
fields: Array[org.apache.spark.sql.types.StructField] = Array(StructField(name,StringType,true), Struc
tField(age,StringType,true))
```

val schema = StructType(fields)

```
scala> val schema = StructType(fields)
schema: org.apache.spark.sql.types.StructType = StructType(StructField(name,StringType,true),StructFie
ld(age,StringType,true))
```

val rowRDD = peopleRDD.map(_.split(“,”)).map(attributed => Row(attributes(0),attributes(1).trim))

```
scala> val rowRDD = peopleRDD.map(_.split(",")).map(attributes => Row(attributes(0), attributes(1).tri
m))
rowRDD: org.apache.spark.rdd.RDD[org.apache.spark.sql.Row] = MapPartitionsRDD[3] at map at <console>:2
7
```

val peopleDF = spark.createDataFrame(rowRDD, schema)

```
scala> val peopleDF = spark.createDataFrame(rowRDD, schema)
peopleDF: org.apache.spark.sql.DataFrame = [name: string, age: string]
```

peopleDF.createOrReplaceTempView(“people”)

```
scala> peopleDF.createOrReplaceTempView("people")
```

val results = spark.sql(“SELECT name FROM people”)

```
scala> val results = spark.sql("SELECT name FROM people")
results: org.apache.spark.sql.DataFrame = [name: string]
```

results.map(attributes => “Name: “ + attributes(0)).show()

```
scala> results.map(attributes => "Name: " + attributes(0)).show()
+-------------+
|        value|
+-------------+
|Name: Michael|
|   Name: Andy|
| Name: Justin|
+-------------+
```

Basic Operations with csv file

val myData = spark.read.format(“csv”).option(“inferSchema”,”true”).option(“header”,”true”).option(“delimeter”,”:”).load(“C:/spark/examples/src/main/resources/people.csv”)

```
scala> val myData = spark.read.format("csv").option("inferSchema","true").option("header"
, true").option("delimeter",":").load("C:/spark/examples/src/main/resources/people.csv")
val myData: org.apache.spark.sql.DataFrame = [name;age;job: string]
```

myData.show()

```
scala> myData.show()
+------------------+
|      name;age;job|
+------------------+
|Jorge;30;Developer|
|  Bob;32;Developer|
+------------------+
```

myData.select($”name”,”$age”).show()

```
scala> myData.select($"name",$"age").show()
+-----+---+
| name|age|
+-----+---+
|Jorge| 30|
|  Bob| 32|
+-----+---+
```

myData.count()

```
scala> myData.count()
val res16: Long = 2
```

myData.count().toDouble

```
scala> myData.count().toDouble
val res17: Double = 2.0
```

NAME: Shaba Khan
ROLL NO. : 31031523039

Practical 03

AIM: Graphx in Apache

CODE & OUTPUT:

import org.apache.spark._
import org.apache.spark.rdd.RDD
import org.apache.spark.graphx._

```
scala> import org.apache.spark._
import org.apache.spark._

scala> import org.apache.spark.rdd.RDD
import org.apache.spark.rdd.RDD

scala> import org.apache.spark.graphx._
import org.apache.spark.graphx._
```

val vertices = Array((1L,("A")),(2L,("B")),(3L,("C")))

```
scala> val vertices = Array((1L,("A")),(2L,("B")),(3L,("C")))
vertices: Array[(Long, String)] = Array((1,A), (2,B), (3,C))
```

val vRDD = sc.parallelize(vertices)

```
scala> val vRDD = sc.parallelize(vertices)
vRDD: org.apache.spark.rdd.RDD[(Long, String)] = ParallelCollectionRDD[0] at parallelize at <console>:
31
```

vRDD.take(1)
vRDD.take(2)

```
scala> vRDD.take(1)
res0: Array[(Long, String)] = Array((1,A))

scala> vRDD.take(2)
res1: Array[(Long, String)] = Array((1,A), (2,B))
```

val edges = Array(Edge(1L,2L,1800),Edge(2L,3L,800),Edge(3L,1L,1400))

```
scala> val edges = Array(Edge(1L,2L,1800),Edge(2L,3L,800),Edge(3L,1L,1400))
edges: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(1,2,1800), Edge(2,3,800), Edge(3,1,1400))
```

val eRDD = sc.parallelize(edges)

```
scala> val eRDD = sc.parallelize(edges)
eRDD: org.apache.spark.rdd.RDD[org.apache.spark.graphx.Edge[Int]] = ParallelCollectionRDD[1] at parall
elize at <console>:31
```

eRDD.take(2)

```
scala> eRDD.take(2)
res2: Array[org.apache.spark.graphx.Edge[Int]] = Array(Edge(1,2,1800), Edge(2,3,800))
```

val nowhere = "nowhere"

```
scala> val nowhere = "nowhere"
nowhere: String = nowhere
```

val graph = Graph(vRDD,eRDD,nowhere)

```
scala> val graph = Graph(vRDD,eRDD,nowhere)
graph: org.apache.spark.graphx.Graph[String,Int] = org.apache.spark.graphx.impl.GraphImpl@dbe577f
```

graph.vertices.collect.foreach(println)
graph.edges.collect.foreach(println)

```
scala> graph.vertices.collect.foreach(println)
(1,A)
(2,B)
(3,C)

scala> graph.edges.collect.foreach(println)
Edge(1,2,1800)
Edge(2,3,800)
Edge(3,1,1400)
```

#to check the no. of airports
val numairports = graph.numVertices

```
scala> val numairports = graph.numVertices
numairports: Long = 3
```

#to check routes
val numroutes = graph.numEdges

```
scala> val numroutes = graph.numEdges
numroutes: Long = 3
```

#routes having distance > 1000
(graph.edges.filter{case Edge(src,dst,prop) => prop>1000}.collect.foreach(println))

```
scala> (graph.edges.filter{case Edge(src,dst,prop) => prop > 1000}.collect.foreach(println))
Edge(1,2,1800)
Edge(3,1,1400)
```

#triplet information
graph.triplets.take(3).foreach(println)

```
scala> graph.triplets.take(3).foreach(println)
((1,A),(2,B),1800)
((2,B),(3,C),800)
((3,C),(1,A),1400)
```

#indegree
val i = graph.inDegrees
i.collect()

```
scala> val i = graph.inDegrees
i: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[21] at RDD at VertexRDD.scala:57

scala> i.collect()
res6: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((1,1), (2,1), (3,1))
```

#outdegrees
val o = graph.ouDegrees
o.collect()

```
scala> val o = graph.outDegrees
o: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[25] at RDD at VertexRDD.scala:57

scala> o.collect()
res7: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((1,1), (2,1), (3,1))
```

#total degree
val t = graph.degrees
t.collect()

```
scala> val t = graph.degrees
t: org.apache.spark.graphx.VertexRDD[Int] = VertexRDDImpl[29] at RDD at VertexRDD.scala:57

scala> t.collect()
res8: Array[(org.apache.spark.graphx.VertexId, Int)] = Array((1,2), (2,2), (3,2))
```

NAME: Shaba Khan
ROLL NO. : 31031523039

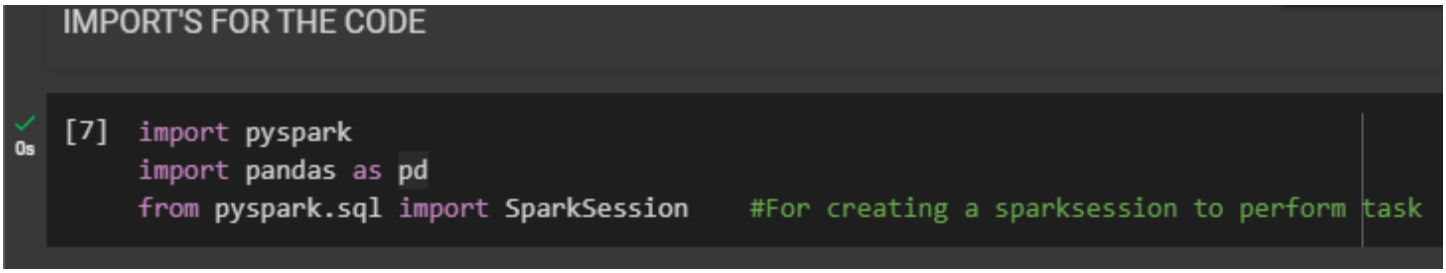
Practical 04

AIM: PySpark

CODE & OUTPUT:

!pip install pyspark

import pyspark
import pandas as pd
from pyspark.sql import SparkSession #For creating a sparksession to perform task



pd.read_csv('/content/sample_data/employees.csv')

READING TEH CSV FILE WHICH YOU HAVE CREATED

```
pd.read_csv('/content/sample_data/employees.csv')
```

	Name	Age	Experience	Salary
0	Alice Smith	28	5	60000
1	Bob Johnson	35	10	85000
2	Charlie Brown	22	1	45000
3	Diana White	30	7	70000
4	Ethan Clark	40	15	95000
5	Fiona Green	27	3	55000
6	George Harris	50	20	120000
7	Hannah Lee	29	6	65000
8	Ian Walker	33	12	80000
9	Julia Adams	25	4	48000
10	Kevin Martin	38	9	72000
11	Laura Thompson	31	8	78000
12	Michael King	45	18	110000
13	Nina Scott	26	2	50000
14	Oscar Turner	37	14	90000
15	Paula Allen	34	11	82000

```
#Start session
spark = SparkSession.builder.appName('Practice').getOrCreate()

#Read dataset and store in variable
df_pyspark = spark.read.option('header','true').csv('/content/sample_data/employees.csv')
df_pyspark.show()
```

```
#Start session
spark = SparkSession.builder.appName('Practice').getOrCreate()

#Read dataset and store in variable
df_pyspark = spark.read.option('header','true').csv('/content/sample_data/employees.csv')
df_pyspark.show()
```

```
+--------------+---+----------+------+
|          Name|Age|Experience|Salary|
+--------------+---+----------+------+
|   Alice Smith| 28|         5| 60000|
|   Bob Johnson| 35|        10| 85000|
| Charlie Brown| 22|         1| 45000|
|   Diana White| 30|         7| 70000|
|   Ethan Clark| 40|        15| 95000|
|   Fiona Green| 27|         3| 55000|
| George Harris| 50|        20|120000|
|    Hannah Lee| 29|         6| 65000|
|    Ian Walker| 33|        12| 80000|
|   Julia Adams| 25|         4| 48000|
|  Kevin Martin| 38|         9| 72000|
|Laura Thompson| 31|         8| 78000|
|  Michael King| 45|        18|110000|
|    Nina Scott| 26|         2| 50000|
|  Oscar Turner| 37|        14| 90000|
|   Paula Allen| 34|        11| 82000|
+--------------+---+----------+------+
```

df_pyspark = spark.read.csv('/content/sample_data/employees.csv', header = True, inferSchema = True)

spark.read.option('header','true').csv('/content/sample_data/employees.csv', inferSchema = True)

#Type of the data
type(df_pyspark)

```
[13] df_pyspark = spark.read.csv('/content/sample_data/employees.csv', header = True, inferSchema = True)

[14] spark.read.option('header','true').csv('/content/sample_data/employees.csv', inferSchema = True)

DataFrame[Name: string, Age: int, Experience: int, Salary: int]

[15] #Type of the data
     type(df_pyspark)

pyspark.sql.dataframe.DataFrame
def __init__(jdf: JavaObject, sql_ctx: Union['SQLContext', 'SparkSession'])
+-------+------+-----------+--------+

Notes
-----
A DataFrame should only be created as described above. It should not be directly
created via using the constructor.
```

#Check Schema
df_pyspark.printSchema()

```
#Check Schema
df_pyspark.printSchema()
```

```
root
 |-- Name: string (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Experience: integer (nullable = true)
 |-- Salary: integer (nullable = true)
```

#First 3 data values
df_pyspark.head(3)

```
[17] #First 3 data values
     df_pyspark.head(3)
```

```
[Row(Name='Alice Smith', Age=28, Experience=5, Salary=60000),
 Row(Name='Bob Johnson', Age=35, Experience=10, Salary=85000),
 Row(Name='Charlie Brown', Age=22, Experience=1, Salary=45000)]
```

#to get name of teh columns
df_pyspark.columns

```
Suggested code may be subject to a license | matthewmcnulty/pyspark-tutorial
#to get name of teh columns
df_pyspark.columns
```

```
['Name', 'Age', 'Experience', 'Salary']
```

#selecting specific columns in the output
df_pyspark.select('Name').show()
df_pyspark.select(['Name','Experience']).show()

```
#selecting specific columns in the output
df_pyspark.select('Name').show()
df_pyspark.select(['Name','Experience']).show()
```

```
+--------------+
|          Name|
+--------------+
|   Alice Smith|
|   Bob Johnson|
| Charlie Brown|
|   Diana White|
|   Ethan Clark|
|   Fiona Green|
| George Harris|
|    Hannah Lee|
|    Ian Walker|
|   Julia Adams|
|  Kevin Martin|
|Laura Thompson|
|  Michael King|
|    Nina Scott|
|  Oscar Turner|
|   Paula Allen|
+--------------+

+--------------+----------+
|          Name|Experience|
+--------------+----------+
|   Alice Smith|         5|
|   Bob Johnson|        10|
| Charlie Brown|         1|
|   Diana White|         7|
|   Ethan Clark|        15|
|   Fiona Green|         3|
| George Harris|        20|
|    Hannah Lee|         6|
|    Ian Walker|        12|
|   Julia Adams|         4|
|  Kevin Martin|         9|
|Laura Thompson|         8|
|  Michael King|        18|
|    Nina Scott|         2|
|  Oscar Turner|        14|
|   Paula Allen|        11|
+--------------+----------+
```

#to check datatypes
df_pyspark.dtypes

```
[21] Suggested code may be subject to a license | thiagodeschamps/spark_learn
#to check datatypes
df_pyspark.dtypes
```

```
[('Name', 'string'), ('Age', 'int'), ('Experience', 'int'), ('Salary', 'int')]
```

#to describe the dataset
df_pyspark.describe().show()

```
#to describe the dataset
df_pyspark.describe().show()
```

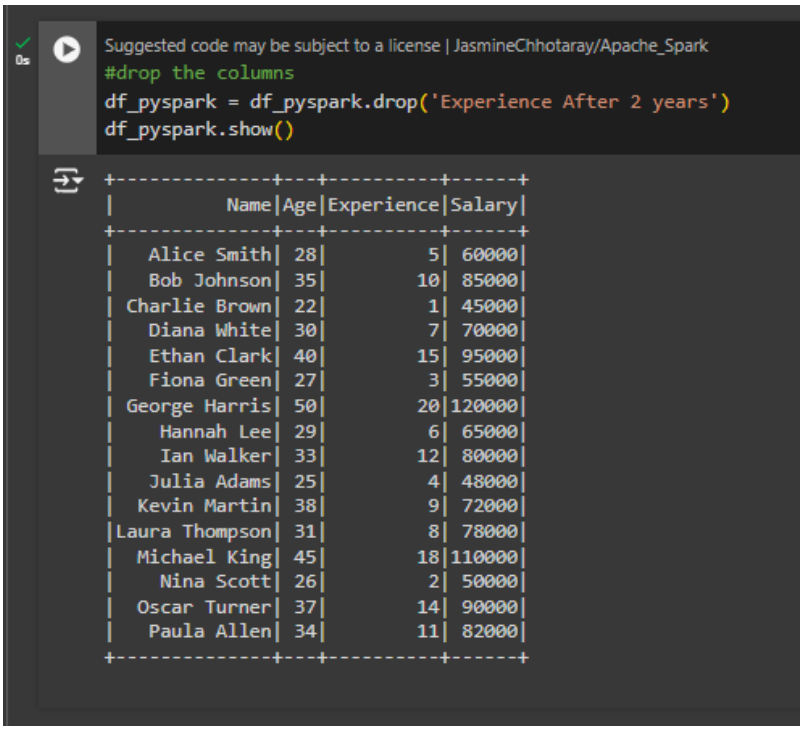
```
+-------+-----------+-----------------+------------------+------------------+
|summary|       Name|              Age|        Experience|            Salary|
+-------+-----------+-----------------+------------------+------------------+
|  count|         16|               16|                16|                16|
|   mean|       NULL|           33.125|            9.0625|           75312.5|
| stddev|       NULL|7.535471672916921|5.6623758264530615|21709.348984558088|
|    min|Alice Smith|               22|                 1|             45000|
|    max|Paula Allen|               50|                20|            120000|
+-------+-----------+-----------------+------------------+------------------+
```

#adding column in teh data frame
df_pyspark = df_pyspark.withColumn('Experience After 2 years', df_pyspark['Experience']+2)
df_pyspark.show()

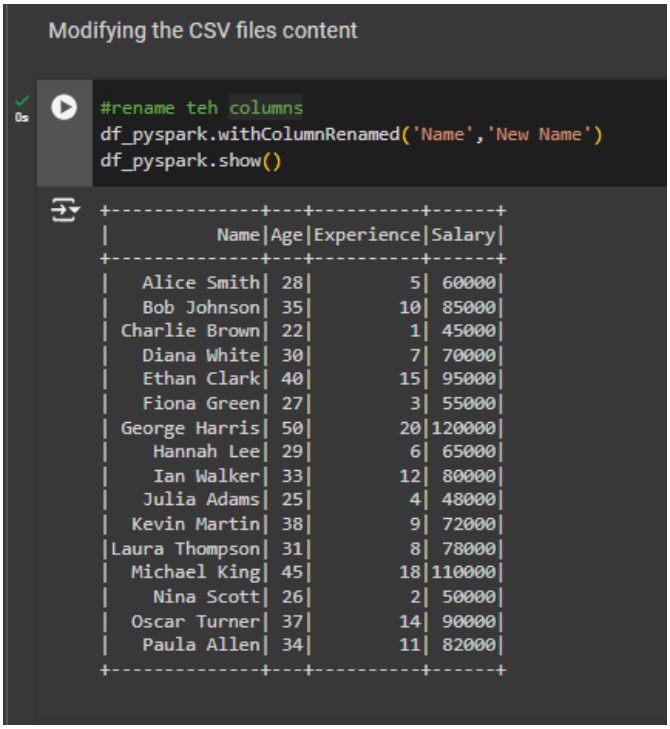
```
[23] Suggested code may be subject to a license | Madhan-sukumar/Pyspark
#adding column in teh data frame
df_pyspark = df_pyspark.withColumn('Experience After 2 years', df_pyspark['Experience']+2)
df_pyspark.show()
```

```
+--------------+---+----------+------+-----------------------+
|          Name|Age|Experience|Salary|Experience After 2 years|
+--------------+---+----------+------+-----------------------+
|   Alice Smith| 28|         5| 60000|                      7|
|   Bob Johnson| 35|        10| 85000|                     12|
| Charlie Brown| 22|         1| 45000|                      3|
|   Diana White| 30|         7| 70000|                      9|
|   Ethan Clark| 40|        15| 95000|                     17|
|   Fiona Green| 27|         3| 55000|                      5|
| George Harris| 50|        20|120000|                     22|
|    Hannah Lee| 29|         6| 65000|                      8|
|    Ian Walker| 33|        12| 80000|                     14|
|   Julia Adams| 25|         4| 48000|                      6|
|  Kevin Martin| 38|         9| 72000|                     11|
|Laura Thompson| 31|         8| 78000|                     10|
|  Michael King| 45|        18|110000|                     20|
|    Nina Scott| 26|         2| 50000|                      4|
|  Oscar Turner| 37|        14| 90000|                     16|
|   Paula Allen| 34|        11| 82000|                     13|
+--------------+---+----------+------+-----------------------+
```

#drop the columns
df_pyspark = df_pyspark.drop('Experience After 2 years')
df_pyspark.show()



#rename the columns
df_pyspark.withColumnRenamed('Name','New Name')
df_pyspark.show()



df_pyspark.na.drop().show()

```
df_pyspark.na.drop().show()
```

```
+--------------+---+----------+------+
|          Name|Age|Experience|Salary|
+--------------+---+----------+------+
|   Alice Smith| 28|         5| 60000|
|   Bob Johnson| 35|        10| 85000|
| Charlie Brown| 22|         1| 45000|
|   Diana White| 30|         7| 70000|
|   Ethan Clark| 40|        15| 95000|
|   Fiona Green| 27|         3| 55000|
| George Harris| 50|        20|120000|
|    Hannah Lee| 29|         6| 65000|
|    Ian Walker| 33|        12| 80000|
|   Julia Adams| 25|         4| 48000|
|  Kevin Martin| 38|         9| 72000|
|Laura Thompson| 31|         8| 78000|
|  Michael King| 45|        18|110000|
|    Nina Scott| 26|         2| 50000|
|  Oscar Turner| 37|        14| 90000|
|   Paula Allen| 34|        11| 82000|
+--------------+---+----------+------+
```

#any==how
df_pyspark.na.drop(how="all").show()

```
Suggested code may be subject to a license | alwinyang91/PySpark-Python
#any==how
df_pyspark.na.drop(how="all").show()
```

```
+--------------+---+----------+------+
|          Name|Age|Experience|Salary|
+--------------+---+----------+------+
|   Alice Smith| 28|         5| 60000|
|   Bob Johnson| 35|        10| 85000|
| Charlie Brown| 22|         1| 45000|
|   Diana White| 30|         7| 70000|
|   Ethan Clark| 40|        15| 95000|
|   Fiona Green| 27|         3| 55000|
| George Harris| 50|        20|120000|
|    Hannah Lee| 29|         6| 65000|
|    Ian Walker| 33|        12| 80000|
|   Julia Adams| 25|         4| 48000|
|  Kevin Martin| 38|         9| 72000|
|Laura Thompson| 31|         8| 78000|
|  Michael King| 45|        18|110000|
|    Nina Scott| 26|         2| 50000|
|  Oscar Turner| 37|        14| 90000|
|   Paula Allen| 34|        11| 82000|
+--------------+---+----------+------+
```

df_pyspark.na.drop(how="any").show()

```
df_pyspark.na.drop(how="any").show()
```

```
+--------------+---+----------+------+
|          Name|Age|Experience|Salary|
+--------------+---+----------+------+
|   Alice Smith| 28|         5| 60000|
|   Bob Johnson| 35|        10| 85000|
| Charlie Brown| 22|         1| 45000|
|   Diana White| 30|         7| 70000|
|   Ethan Clark| 40|        15| 95000|
|   Fiona Green| 27|         3| 55000|
| George Harris| 50|        20|120000|
|    Hannah Lee| 29|         6| 65000|
|    Ian Walker| 33|        12| 80000|
|   Julia Adams| 25|         4| 48000|
|  Kevin Martin| 38|         9| 72000|
|Laura Thompson| 31|         8| 78000|
|  Michael King| 45|        18|110000|
|    Nina Scott| 26|         2| 50000|
|  Oscar Turner| 37|        14| 90000|
|   Paula Allen| 34|        11| 82000|
+--------------+---+----------+------+
```

df_pyspark.na.drop(how="any", thresh = 2).show()

```
df_pyspark.na.drop(how="any", thresh = 2).show()
```

```
+--------------+---+----------+------+
|          Name|Age|Experience|Salary|
+--------------+---+----------+------+
|   Alice Smith| 28|         5| 60000|
|   Bob Johnson| 35|        10| 85000|
| Charlie Brown| 22|         1| 45000|
|   Diana White| 30|         7| 70000|
|   Ethan Clark| 40|        15| 95000|
|   Fiona Green| 27|         3| 55000|
| George Harris| 50|        20|120000|
|    Hannah Lee| 29|         6| 65000|
|    Ian Walker| 33|        12| 80000|
|   Julia Adams| 25|         4| 48000|
|  Kevin Martin| 38|         9| 72000|
|Laura Thompson| 31|         8| 78000|
|  Michael King| 45|        18|110000|
|    Nina Scott| 26|         2| 50000|
|  Oscar Turner| 37|        14| 90000|
|   Paula Allen| 34|        11| 82000|
+--------------+---+----------+------+
```

df_pyspark.na.drop(how="any", subset = ['Experience']).show()

```
[32] df_pyspark.na.drop(how="any", subset = ['Experience']).show()
```

```
+--------------+---+----------+------+
|          Name|Age|Experience|Salary|
+--------------+---+----------+------+
|   Alice Smith| 28|         5| 60000|
|   Bob Johnson| 35|        10| 85000|
| Charlie Brown| 22|         1| 45000|
|   Diana White| 30|         7| 70000|
|   Ethan Clark| 40|        15| 95000|
|   Fiona Green| 27|         3| 55000|
| George Harris| 50|        20|120000|
|    Hannah Lee| 29|         6| 65000|
|    Ian Walker| 33|        12| 80000|
|   Julia Adams| 25|         4| 48000|
|  Kevin Martin| 38|         9| 72000|
|Laura Thompson| 31|         8| 78000|
|  Michael King| 45|        18|110000|
|    Nina Scott| 26|         2| 50000|
|  Oscar Turner| 37|        14| 90000|
|   Paula Allen| 34|        11| 82000|
+--------------+---+----------+------+
```

#filling the missing value
df_pyspark.na.fill('Missing Values').show()

```
[33] Suggested code may be subject to a license | JasmineChhotaray/Apache_Spark
#filling the missing value
df_pyspark.na.fill('Missing Values').show()
```

```
+--------------+---+----------+------+
|          Name|Age|Experience|Salary|
+--------------+---+----------+------+
|   Alice Smith| 28|         5| 60000|
|   Bob Johnson| 35|        10| 85000|
| Charlie Brown| 22|         1| 45000|
|   Diana White| 30|         7| 70000|
|   Ethan Clark| 40|        15| 95000|
|   Fiona Green| 27|         3| 55000|
| George Harris| 50|        20|120000|
|    Hannah Lee| 29|         6| 65000|
|    Ian Walker| 33|        12| 80000|
|   Julia Adams| 25|         4| 48000|
|  Kevin Martin| 38|         9| 72000|
|Laura Thompson| 31|         8| 78000|
|  Michael King| 45|        18|110000|
|    Nina Scott| 26|         2| 50000|
|  Oscar Turner| 37|        14| 90000|
|   Paula Allen| 34|        11| 82000|
+--------------+---+----------+------+
```

#add imputation columns to df
imputer.fit(df_pyspark).transform(df_pyspark).show()

```
#add imputation columns to df
imputer.fit(df_pyspark).transform(df_pyspark).show()
```

```
+--------------+---+----------+------+-----------+-----------------+--------------+
|          Name|Age|Experience|Salary|Age_imputed|Experience_imputed|Salary_imputed|
+--------------+---+----------+------+-----------+-----------------+--------------+
|   Alice Smith| 28|         5| 60000|         28|                5|         60000|
|   Bob Johnson| 35|        10| 85000|         35|               10|         85000|
| Charlie Brown| 22|         1| 45000|         22|                1|         45000|
|   Diana White| 30|         7| 70000|         30|                7|         70000|
|   Ethan Clark| 40|        15| 95000|         40|               15|         95000|
|   Fiona Green| 27|         3| 55000|         27|                3|         55000|
| George Harris| 50|        20|120000|         50|               20|        120000|
|    Hannah Lee| 29|         6| 65000|         29|                6|         65000|
|    Ian Walker| 33|        12| 80000|         33|               12|         80000|
|   Julia Adams| 25|         4| 48000|         25|                4|         48000|
|  Kevin Martin| 38|         9| 72000|         38|                9|         72000|
|Laura Thompson| 31|         8| 78000|         31|                8|         78000|
|  Michael King| 45|        18|110000|         45|               18|        110000|
|    Nina Scott| 26|         2| 50000|         26|                2|         50000|
|  Oscar Turner| 37|        14| 90000|         37|               14|         90000|
|   Paula Allen| 34|        11| 82000|         34|               11|         82000|
+--------------+---+----------+------+-----------+-----------------+--------------+
```

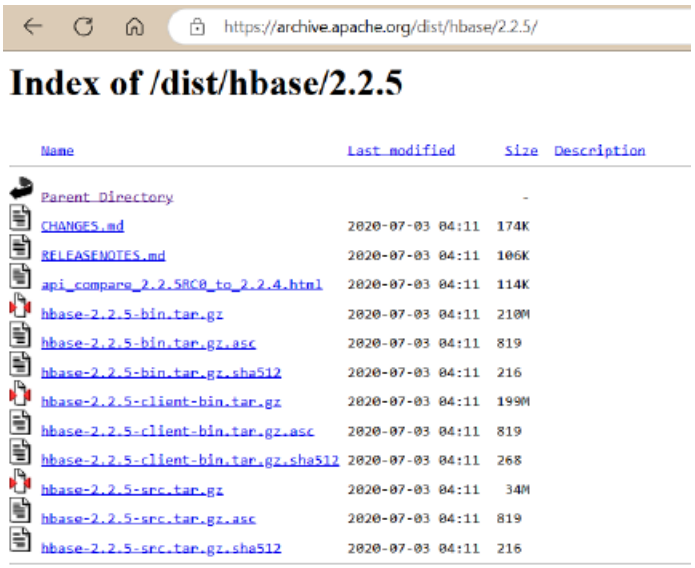
NAME: Shaba Khan
ROLL NO. : 31031523039

Practical 05

AIM: Install HBase

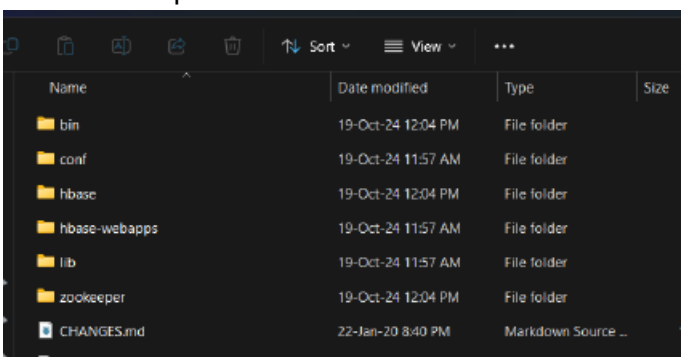
STEPS:

Step 1: Download HBase: Get the HBase binary file from the official website.



Step 2: Create Directory & Extract Files:

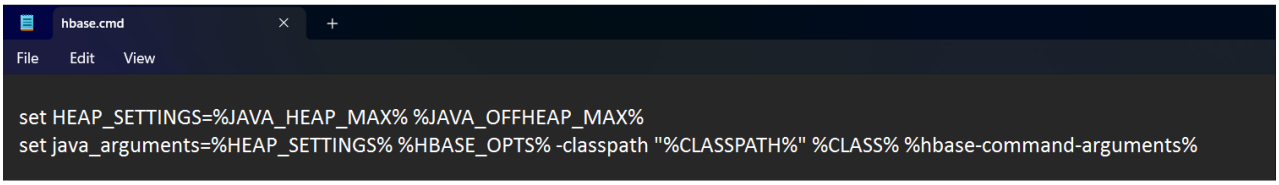
- Make a new folder named hbasesetup in the C: drive. Extract HBase files into this folder.



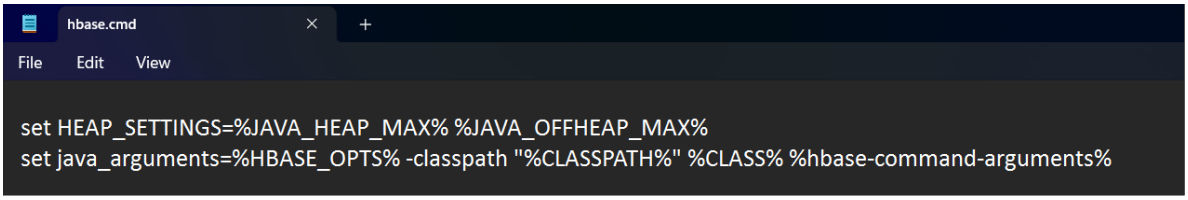
- Inside hbasesetup, create two additional folders named hbase and zookeeper.

Step 3: Edit hbase.cmd:

- Open the bin folder.
- Find hbase.cmd and open it with Notepad.



- Locate the java_arguments line and remove the %HEAP_SETTINGS% variable.



Step 4: Edit hbase-env.cmd:

- Go to the conf folder and open hbase-env.cmd in Notepad.
- Add the following lines:

```
set JAVA_HOME=C:\Progra~1\Java\jdk1.8.0_202
set HBASE_CLASSPATH=%HBASE_HOME%\lib\client-facing-thirdparty\*
set HBASE_HEAPSIZE=8000
set HBASE_OPTS="-XX:+UseConcMarkSweepGC" "-Djava.net.preferIPv4Stack=true"
set SERVER_GC_OPTS="-verbose:gc" "-XX:+PrintGCDetails" "-XX:+PrintGCDateStamps"
%HBASE_GC_OPTS%
set HBASE_USE_GC_LOGFILE=true
set HBASE_JMX_BASE="-Dcom.sun.management.jmxremote.ssl=false"
"-Dcom.sun.management.jmxremote.authenticate=false"
set HBASE_MASTER_OPTS=%HBASE_JMX_BASE% "-Dcom.sun.management.jmxremote.port=10101"
set HBASE_REGIONSERVER_OPTS=%HBASE_JMX_BASE%
"-Dcom.sun.management.jmxremote.port=10102"
set HBASE_THRIFT_OPTS=%HBASE_JMX_BASE% "-Dcom.sun.management.jmxremote.port=10103"
set HBASE_ZOOKEEPER_OPTS=%HBASE_JMX_BASE%
-Dcom.sun.management.jmxremote.port=10104"
set HBASE_REGIONSERVERS=%HBASE_HOME%\conf\regionservers
set HBASE_LOG_DIR=%HBASE_HOME%\logs
set HBASE_IDENT_STRING=%USERNAME%
set HBASE_MANAGES_ZK=true
```

hbase-env.cmd

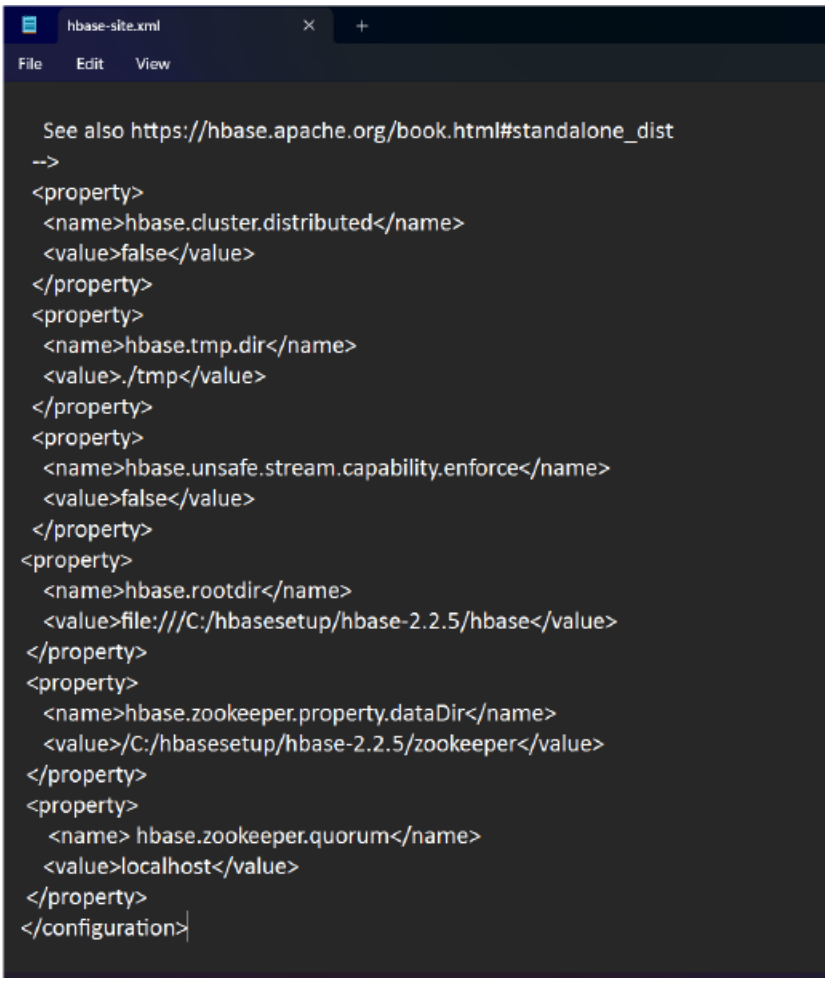
File Edit View

```
@rem The java implementation to use.  Java 1.8+ required.
@rem set JAVA_HOME=c:\apps\java
set JAVA_HOME=C:\Progra~1\Java\jdk1.8.0_202
set HBASE_CLASSPATH=%HBASE_HOME%\lib\client-facing-thirdparty\*
set HBASE_HEAPSIZE=8000
set HBASE_OPTS="-XX:+UseConcMarkSweepGC" "-Djava.net.preferIPv4Stack=true"
set SERVER_GC_OPTS="-verbose:gc" "-XX:+PrintGCDetails" "-XX:+PrintGCDateStamps" %HBASE_GC_OPTS%
set HBASE_USE_GC_LOGFILE=true

set HBASE_JMX_BASE="-Dcom.sun.management.jmxremote.ssl=false" "-Dcom.sun.management.jmxremote.authenticate=false"

set HBASE_MASTER_OPTS=%HBASE_JMX_BASE% "-Dcom.sun.management.jmxremote.port=10101"
set HBASE_REGIONSERVER_OPTS=%HBASE_JMX_BASE% "-Dcom.sun.management.jmxremote.port=10102"
set HBASE_THRIFT_OPTS=%HBASE_JMX_BASE% "-Dcom.sun.management.jmxremote.port=10103"
set HBASE_ZOOKEEPER_OPTS=%HBASE_JMX_BASE% -Dcom.sun.management.jmxremote.port=10104"
set HBASE_REGIONSERVERS=%HBASE_HOME%\conf\regionservers
set HBASE_LOG_DIR=%HBASE_HOME%\logs
set HBASE_IDENT_STRING=%USERNAME%
set HBASE_MANAGES_ZK=true
```

Step 5: Edit hbase-site.xml:

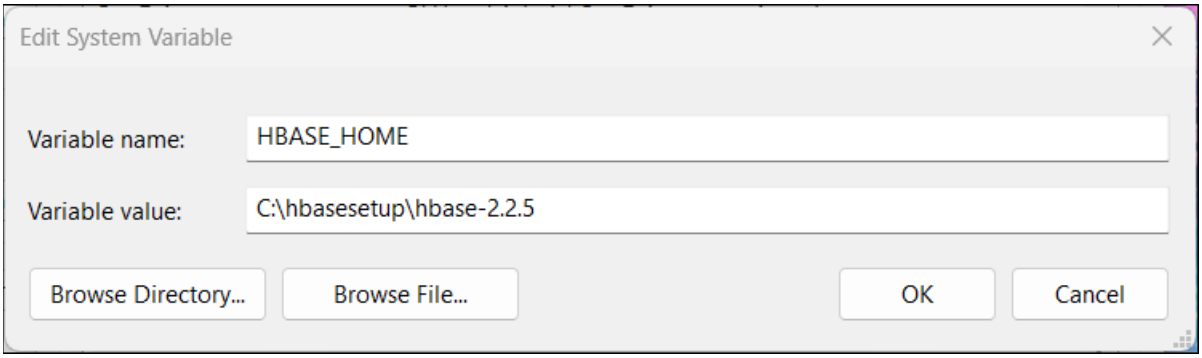
- In the conf folder, open hbase-site.xml in Notepad.
- Add the following properties right after the last property tag:

```
<property>
<name>hbase.rootdir</name>
<value>file:///C:/hbasesetup/hbase-2.2.5/hbase</value>
</property>
<property>
<name>hbase.zookeeper.property.dataDir</name>
<value>/C:/hbasesetup/hbase-2.2.5/zookeeper</value>
</property>
<property>
<name> hbase.zookeeper.quorum</name>
<value>localhost</value>
</property>
```



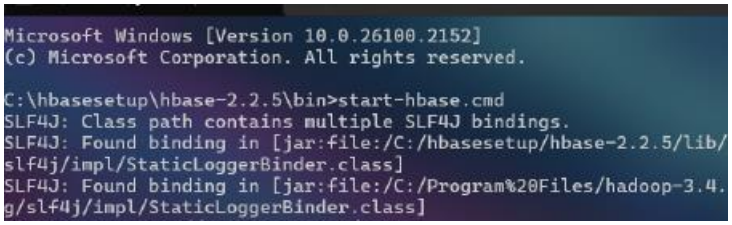
Step 6: Set Environment Variables:

- Configure the HBase environment variables and add them to the system's path.



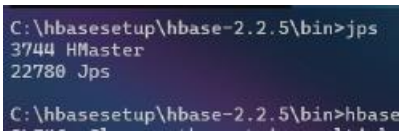
Step 7: Start HBase:

- Open Command Prompt and navigate to the bin folder in HBase.
- Run the command start-hbase.cmd to launch HBase.



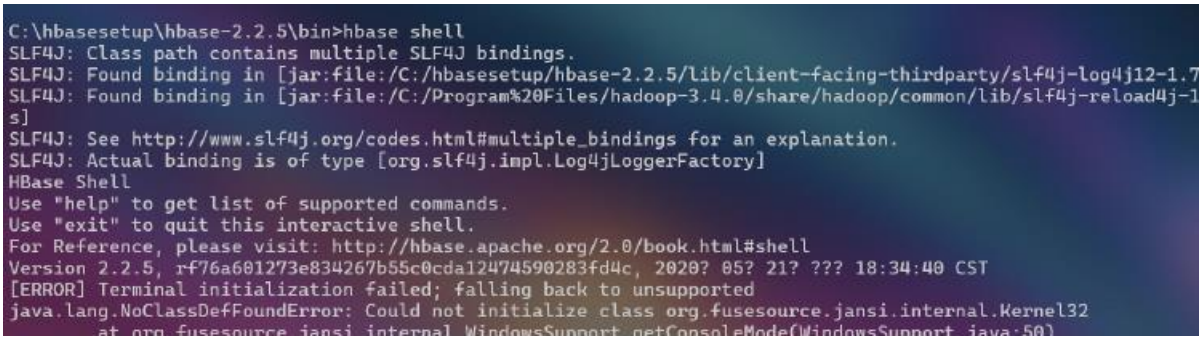
Step 8: Verify HMaster:

- Use the jps command to check if HMaster is running.



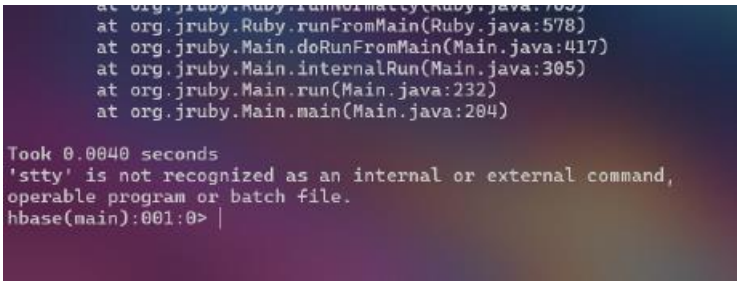
Step 9: Start HBase Shell:

- Launch the HBase shell by typing hbase shell. The initial startup might take some time.



Step 10: Ignore Warnings:

- Once the shell starts, ignore any warnings that appear.



******