

# **Faculty of Engineering**

**Assignment 1- Semester 7: May 2025**

**Module Name: Cloud Computing Module**

**Number: EC7205**

**Assignment Title: Large-Scale Data Analysis Using MapReduce**

**Team Members:**

1. Dissanayaka R.P.L.M      EG/2020/3909
2. Dissanayake D.K.R.C.K      EG/2020/3910
3. Balasooriya B.A.L.M      EG/2020/3838

**Deadline: 07<sup>th</sup> June 2025**

# Contents

1	Objective .....	4
2	Dataset Selection.....	4
2.1	Dataset Description .....	4
2.1.1	Dataset Columns .....	4
2.2	Dataset Justification .....	5
3	Environment Setup.....	6
3.1	WSL & Ubuntu .....	6
3.2	SSH Setup .....	7
3.3	Install Java jdk 8 .....	7
3.4	Hadoop Installation .....	8
3.5	Configuration Files .....	9
3.6	Environment Variables in ~/.bashrc .....	9
3.7	Start Hadoop .....	10
4	MapReduce Task Implementation.....	11
4.1	Task Description .....	11
4.2	Algorithm Design.....	11
4.3	Source Code Summary.....	12
5	Execution and Testing .....	14
5.1	Upload Dataset to HDFS.....	14
5.2	Run MapReduce Job .....	14
5.3	Output Sample.....	15
6	Results and Interpretation .....	16
7	Hadoop Web Interfaces .....	17

# List of Figures

Figure 2.1 : About the Wine review dataset from Keggles.....	5
Figure 3.1 : WSL Installation via PowerShell.....	6
Figure 3.2 : Ubuntu package update .....	7
Figure 3.3 : Command for remove and install OpenSSH .....	7
Figure 3.4 : Java installation command line .....	8
Figure 3.5 : Java version check command .....	8
Figure 3.6 : Download and exact Hadoop.....	8
Figure 3.7 : Navigating to Hadoop Configuration File core-site.xml using VS Code.....	9
Figure 3.8 : Open .bashrc file.....	9
Figure 3.9 : bashrc file .....	9
Figure 3.10 : Code for source.....	10
Figure 3.11 : Format the HDFS NameNode .....	10
Figure 3.12 : Start the HDFS daemons .....	10
Figure 3.13 : Start the YARN daemons.....	10
Figure 3.14 : Output of the jps command .....	11
Figure 4.1 : AveragePricePerVariety.java file .....	12
Figure 4.2 : AveragePriceReducer.java file code .....	12
Figure 4.3 : PriceMapper.java file code .....	13
Figure 5.1 : Creating HDFS directory and uploading a CSV file in Hadoop .....	14
Figure 5.2 : Hadoop MapReduce job calculating average wine prices per variety .....	14
Figure 5.3 : Command-line interface .....	15
Figure 5.4 : Hadoop MapReduce output: Average wine prices by variety.....	15
Figure 6.1 : Output terminal of wine varieties and their calculated average prices.....	16
Figure 7.1 : Hadoop web interface (NameNode).....	17
Figure 7.2 : Hadoop web interface (ResourceManager).....	18

# 1 Objective

This project aims to analyze a large wine review dataset using Hadoop MapReduce to extract meaningful insights. Specifically, it calculates the average price per wine variety, filtering out incomplete or malformed records. The solution is implemented in Java using the Hadoop MapReduce programming model.

## 2 Dataset Selection

### 2.1 Dataset Description

This project utilizes the Wine Reviews dataset from Kaggle, which contains over 130,000 wine reviews including information such as variety, price, description, region, and rating. The dataset is valuable for analysis due to its mix of structured (e.g., price, points) and unstructured (e.g., description) data. For our project, the focus is on extracting the average price per wine variety using Hadoop MapReduce.

- **Dataset Name:** Wine Reviews
- **Source:** Kaggle (<https://www.kaggle.com/datasets/zynicide/wine-reviews>)
- **Link:** Wine Reviews Dataset
- **File Format:** CSV
- **Size:** 130,000+ reviews
- **License:** CC BY-NC-SA 4.0

#### 2.1.1 Dataset Columns

- country
- description
- designation
- points
- price
- province
- region\_1
- region\_2
- variety (target group)

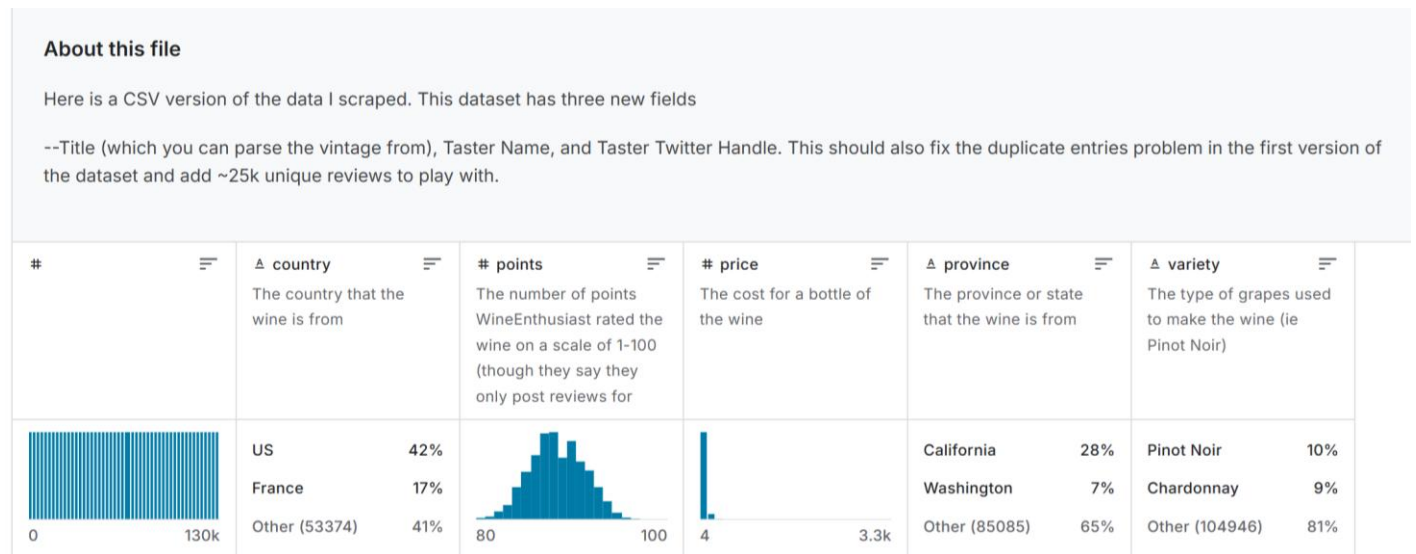


Figure 2.1 : About the Wine review dataset from Kaggle

## 2.2 Dataset Justification

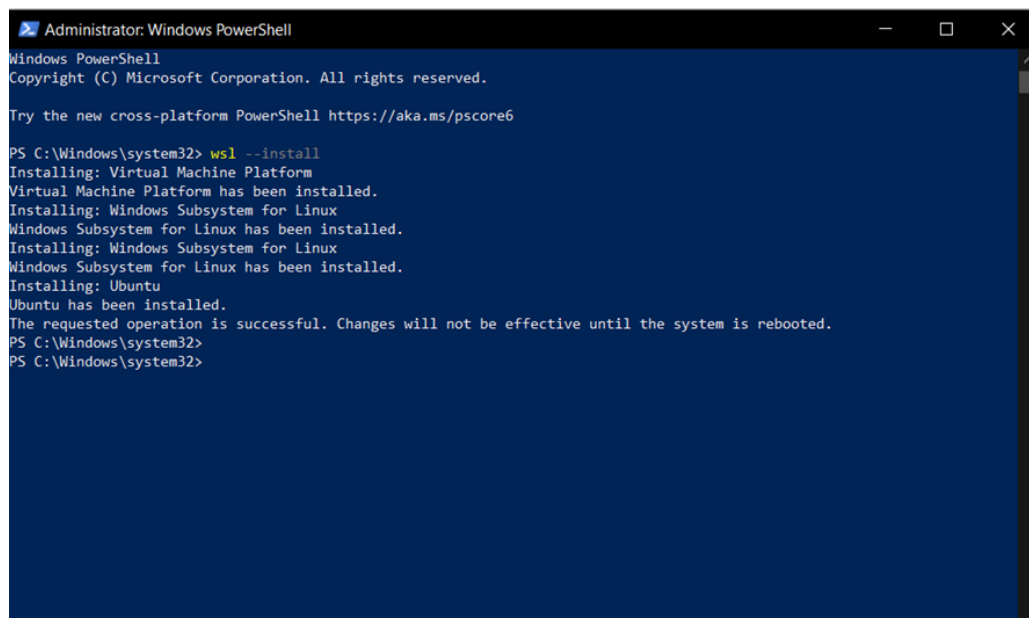
The Wine Reviews dataset is an ideal candidate for distributed data processing with Hadoop MapReduce due to its large volume and diverse structure. It contains a blend of categorical, numerical, and textual data, allowing us to explore a variety of analytical goals. In particular, the inclusion of wine variety and price information enables grouped analysis and aggregation operations that are naturally parallelizable. Processing this data with MapReduce not only improves scalability but also demonstrates how Hadoop can efficiently handle real-world datasets that include missing or inconsistent entries. By computing the average price for each wine variety, this project delivers practical insights that could be valuable to wine retailers, sommeliers, and consumers alike, especially in identifying pricing patterns across different types of wines.

### 3 Environment Setup

Setting up a Hadoop environment for Java-based MapReduce involves configuring several components including WSL (Windows Subsystem for Linux), Java, SSH, and Hadoop binaries. Below is a detailed breakdown of each step required to get the project running smoothly.

#### 3.1 WSL & Ubuntu

Windows Subsystem for Linux (WSL) allows us to run a Linux environment directly on Windows. For this project, we use Ubuntu as the Linux distribution under WSL. This setup enables seamless integration between Linux-native tools and Hadoop. Installation command has shown on below Figure 3.1.

A screenshot of a Windows PowerShell terminal window titled "Administrator: Windows PowerShell". The terminal shows the output of the command "wsl --install". The output indicates that the Virtual Machine Platform has been installed, followed by the Windows Subsystem for Linux, and finally Ubuntu. A message at the end states: "The requested operation is successful. Changes will not be effective until the system is rebooted." The prompt "PS C:\Windows\system32>" is visible at the bottom.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Windows\system32> wsl --install
Installing: Virtual Machine Platform
Virtual Machine Platform has been installed.
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.
Installing: Ubuntu
Ubuntu has been installed.
The requested operation is successful. Changes will not be effective until the system is rebooted.
PS C:\Windows\system32>
```

Figure 3.1 : WSL Installation via PowerShell

After rebooting, Ubuntu is opened and update packages. This ensures machine have the latest system updates and dependencies. These commands have shown by Figure 3.2.

```

Welcome to Ubuntu 24.04.2 LTS (GNU/Linux 6.6.87.1-microsoft-standard-WSL2 x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Thu May 29 11:35:27 +0530 2025

System load:  0.0               Processes:    34
Usage of /:   0.1% of 1006.85GB Users logged in: 0
Memory usage: 10%              IPv4 address for eth0: 172.23.139.137
Swap usage:   0%

This message is shown once a day. To disable it please create the
/home/malindu/.hushlogin file.
malindu@DESKTOP-S16UMS3:~$ sudo apt update
sudo apt install openjdk-11-jdk -y
java -version
[sudo] password for malindu:
Sorry, try again.
[sudo] password for malindu:
Ign:1 http://archive.ubuntu.com/ubuntu noble InRelease
Get:2 http://archive.ubuntu.com/ubuntu noble-updates InRelease [126 kB]
Get:3 http://archive.ubuntu.com/ubuntu noble-backports InRelease [126 kB]
Hit:1 http://archive.ubuntu.com/ubuntu noble InRelease
Get:4 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Packages [1107 kB]
Get:5 http://archive.ubuntu.com/ubuntu noble-updates/main Translation-en [235 kB]
Get:6 http://archive.ubuntu.com/ubuntu noble-updates/main amd64 Components [161 kB]

```

Figure 3.2 : Ubuntu package update

## 3.2 SSH Setup

Hadoop uses SSH for managing cluster nodes even in a pseudo distributed single-node setup. It's essential that passwordless SSH is set up so Hadoop daemons can communicate effectively during start-up and task execution.

Before installing a new version of OpenSSH, it's a good idea to remove any existing version that might already be installed. This helps prevent potential issues caused by version conflicts or mismatched components.

```

sudo apt remove openssh-server
sudo apt install openssh-server
ssh-keygen -t rsa
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
sudo service ssh start

```

Figure 3.3 : Command for remove and install OpenSSH

## 3.3 Install Java jdk 8

Hadoop requires Java to run all core components, including NameNode, DataNode, and YARN services. OpenJDK 8 has used as it is stable and compatible with Hadoop 3.x.

Install Java using following command has mentioned by Figure 3.4

```
malindu@DESKTOP-S16UMS3:~$ sudo apt --assume-yes install openjdk-8-jre-headless
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openjdk-8-jre-headless is already the newest version (8u452-ga~us1-0ubuntu1~24.04).
openjdk-8-jre-headless set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 111 not upgraded.
```

Figure 3.4 : Java installation command line

After installation verify the installation with using java -version command

```
malindu@DESKTOP-S16UMS3:~$ java -version
openjdk version "1.8.0_452"
OpenJDK Runtime Environment (build 1.8.0_452-8u452-ga~us1-0ubuntu1~24.04-b09)
OpenJDK 64-Bit Server VM (build 25.452-b09, mixed mode)
malindu@DESKTOP-S16UMS3:~$ which java
/usr/bin/java
```

Figure 3.5 : Java version check command

### 3.4 Hadoop Installation

Download and extract Hadoop binaries using the commands below Figure 3.6. We use version 3.3.6 for this project due to its stability and compatibility with recent Java versions.

```
malindu@DESKTOP-S16UMS3: ~
System load: 0.0          Processes: 60
Usage of /: 0.1% of 1006.85GB  Users logged in: 0
Memory usage: 10%          IPv4 address for eth0: 172.23.139.137
Swap usage: 0%

This message is shown once a day. To disable it please create the
/home/malindu/.hushlogin file.
malindu@DESKTOP-S16UMS3:~$ wget https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
p-3.3.6.tar.gz
mv hadoop-3.3.6 ~/hadoop
--2025-06-02 13:43:52-- https://downloads.apache.org/hadoop/common/hadoop-3.3.6/hadoop-3.3.6.tar.gz
Resolving downloads.apache.org (downloads.apache.org)... 135.181.214.104, 88.99.208.237, 2a01:4f9:3a:2c57::2, ...
Connecting to downloads.apache.org (downloads.apache.org)|135.181.214.104|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 730107476 (696M) [application/x-gzip]
Saving to: 'hadoop-3.3.6.tar.gz.1'

hadoop-3.3.6.tar.gz.1      100%[=====] 696.28M  4.39MB/s   in 27m 8s

2025-06-02 14:11:01 (438 KB/s) - 'hadoop-3.3.6.tar.gz.1' saved [730107476/730107476]

malindu@DESKTOP-S16UMS3:~$ tar -xzf hadoop-3.3.6.tar.gz

gzip: stdin: unexpected end of file
tar: Unexpected EOF in archive
tar: Unexpected EOF in archive
tar: Error is not recoverable: exiting now
malindu@DESKTOP-S16UMS3:~$ mv hadoop-3.3.6 ~/hadoop
malindu@DESKTOP-S16UMS3:~$
```

Figure 3.6 : Download and exact Hadoop

Ensure the Hadoop folder is accessible and properly located inside your home directory.



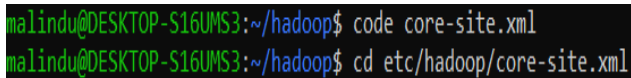
### 3.5 Configuration Files

Hadoop requires configuration files to define file system properties, replication settings, job execution framework, and resource manager communication.

Files to configure:

- **core-site.xml**: Defines the NameNode (HDFS) URI
- **hdfs-site.xml**: Sets HDFS block size, replication, and storage directories
- **mapred-site.xml**: Configures MapReduce to run on YARN
- **yarn-site.xml**: Specifies ResourceManager hostname and NodeManager services

We can edit above things using VS Code or nano. For example,

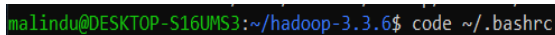


```
malindu@DESKTOP-S16UMS3:~/hadoop$ code core-site.xml
malindu@DESKTOP-S16UMS3:~/hadoop$ cd etc/hadoop/core-site.xml
```

Figure 3.7 : Navigating to Hadoop Configuration File core-site.xml using VS Code

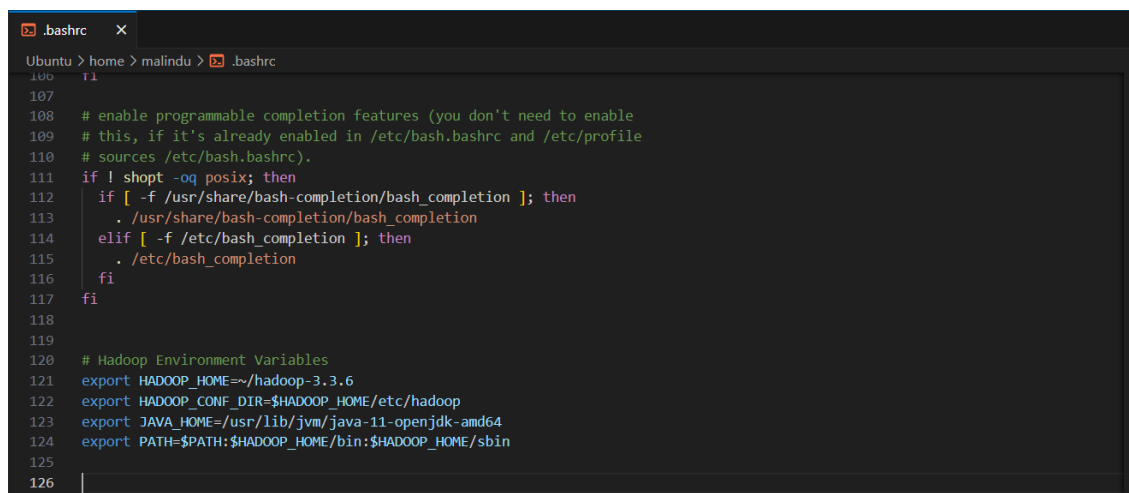
### 3.6 Environment Variables in ~/.bashrc

To allow terminal wide access to Hadoop commands, should add the following environment variables to .bashrc file.



```
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$ code ~/.bashrc
```

Figure 3.8 : Open .bashrc file



```
.bashrc
106
107
108 # enable programmable completion features (you don't need to enable
109 # this, if it's already enabled in /etc/bash.bashrc and /etc/profile
110 # sources /etc/bash.bashrc).
111 if ! shopt -oq posix; then
112   if [ -f /usr/share/bash-completion/bash_completion ]; then
113     . /usr/share/bash-completion/bash_completion
114   elif [ -f /etc/bash_completion ]; then
115     . /etc/bash_completion
116   fi
117 fi
118
119
120 # Hadoop Environment Variables
121 export HADOOP_HOME=~/hadoop-3.3.6
122 export HADOOP_CONF_DIR=$HADOOP_HOME/etc/hadoop
123 export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
124 export PATH=$PATH:$HADOOP_HOME/bin:$HADOOP_HOME/sbin
125
126 |
```

Figure 3.9 : bashrc file

After editing, should apply the changes.

```
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$ source ~/.bashrc
```

Figure 3.10 : Code for source

This enables the use of commands like hdfs, yarn, start-dfs.sh, and others globally in the terminal.

### 3.7 Start Hadoop

To initialize and start the Hadoop file system and YARN services should follow these steps.

1. Format the HDFS NameNode (only required for the first-time setup)

```
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$ bin/hdfs namenode -format
WARNING: /home/malindu/hadoop-3.3.6/logs does not exist. Creating.
2025-06-03 00:31:56,947 INFO namenode.NameNode: STARTUP_MSG:
/*****
STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = DESKTOP-S16UMS3.localdomain/127.0.1.1
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 3.3.6
STARTUP_MSG: classpath = /home/malindu/hadoop-3.3.6/etc/hadoop:/home/malindu/hadoop-3.3.6/share/hadoop/common/lib/nett
y-codec-dns-4.1.89.Final.jar:/home/malindu/hadoop-3.3.6/share/hadoop/common/lib/curator-recipes-5.2.0.jar:/home/malindu/
hadoop-3.3.6/share/hadoop/common/lib/hadoop-shaded-protobuf_3_7-1.1.1.jar:/home/malindu/hadoop-3.3.6/share/hadoop/common
/lib/netty-handler-ssl-ocsp-4.1.89.Final.jar:/home/malindu/hadoop-3.3.6/share/hadoop/common/lib/jackson-mapper-asl-1.9.1
3.jar:/home/malindu/hadoop-3.3.6/share/hadoop/common/lib/hadoop-shaded-guava-1.1.1.jar:/home/malindu/hadoop-3.3.6/share/
hadoop/common/lib/netty-transport-native-epoll-4.1.89.Final-linux-x86_64.jar:/home/malindu/hadoop-3.3.6/share/hadoop/com
mon/lib/httpclient-4.5.13.jar:/home/malindu/hadoop-3.3.6/share/hadoop/common/lib/kerb-util-1.0.1.jar:/home/malindu/had
```

Figure 3.11 : Format the HDFS NameNode

This sets up the directory structure and initializes HDFS metadata.

2. Start the HDFS daemons

```
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$ sbin/start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [DESKTOP-S16UMS3]
DESKTOP-S16UMS3: Warning: Permanently added 'desktop-s16ums3' (ED25519) to the list of known hosts.
```

Figure 3.12 : Start the HDFS daemons

We should see output that includes both NameNode and DataNode.

3. Start the YARN daemons

```
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$ sbin/start-yarn.sh
Starting resourcemanager
Starting nodemanagers
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$
```

Figure 3.13 : Start the YARN daemons

This command starts the NameNode, DataNode, ResourceManager and NodeManager. Verify they are running with following command (Figure 3.14).

```
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$ jps
1345 NodeManager
822 DataNode
1001 SecondaryNameNode
1228 ResourceManager
1710 Jps
```

Figure 3.14 : Output of the jps command

Once everything is running, you can open the Hadoop Web UIs:

- HDFS NameNode: <http://localhost:9870>
- YARN ResourceManager: <http://localhost:8088>

## 4 MapReduce Task Implementation

MapReduce is a programming model for processing large datasets with a distributed algorithm on a Hadoop cluster. In this project, the MapReduce job is implemented in Java to compute the average wine price grouped by variety. It includes a Mapper to extract key-value pairs, and a Reducer to aggregate and compute averages.

### 4.1 Task Description

The main objective of this task is to calculate the average wine price per variety using the dataset. Each row of the CSV contains details such as price and wine variety. Rows with missing or invalid prices or varieties are skipped. This ensures only clean and usable data is used in calculations.

The final output is a list of wine varieties along with their respective average prices. This result can help consumers and businesses understand which varieties are priced higher or lower on average.

### 4.2 Algorithm Design

- **Mapper Function:** Reads each line of the dataset and splits it by commas. Extracts the 'variety' and 'price' columns. Emits the wine variety as the key and its price as the value. Skips malformed rows with missing or non-numeric prices.
- **Reducer Function:** For each unique wine variety key, receives a list of prices. It computes the sum of prices and counts how many prices were received. Then it calculates the average by dividing the sum by the count and emits the variety with its average price.

This two-step process ensures distributed, parallel processing of large datasets with accurate aggregation.

## 4.3 Source Code Summary

- **Language:** Java
- **Source code:** [https://github.com/MalinduDissanayaka/wine\\_review\\_analyzer\\_project.git](https://github.com/MalinduDissanayaka/wine_review_analyzer_project.git)
- **Files:**
  1. AveragePricePerVariety.java (Driver)
  2. PriceMapper.java
  3. AveragePriceReducer.java

```
1  import org.apache.hadoop.conf.Configuration;
2  import org.apache.hadoop.fs.Path;
3  import org.apache.hadoop.io.DoubleWritable;
4  import org.apache.hadoop.io.Text;
5  import org.apache.hadoop.mapreduce.Job;
6  import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
7  import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
8
9  public class AveragePricePerVariety {
10     public static void main(String[] args) throws Exception {
11         Configuration conf = new Configuration();
12         Job job = Job.getInstance(conf, "Average Price per Wine Variety");
13
14         job.setJarByClass(AveragePricePerVariety.class);
15         job.setMapperClass(PriceMapper.class);
16         job.setReducerClass(AveragePriceReducer.class);
17
18         job.setMapOutputKeyClass(Text.class);
19         job.setMapOutputValueClass(DoubleWritable.class);
20         job.setOutputKeyClass(Text.class);
21         job.setOutputValueClass(DoubleWritable.class);
22
23         FileInputFormat.addInputPath(job, new Path(args[0]));
24         FileOutputFormat.setOutputPath(job, new Path(args[1]));
25
26         System.exit(job.waitForCompletion(true) ? 0 : 1);
27     }
28 }
```

Figure 4.1 : AveragePricePerVariety.java file

```
1  import org.apache.hadoop.io.DoubleWritable;
2  import org.apache.hadoop.io.Text;
3  import org.apache.hadoop.mapreduce.Reducer;
4  import java.io.IOException;
5
6  public class AveragePriceReducer extends Reducer<Text, DoubleWritable, Text, DoubleWritable> {
7     private final DoubleWritable averagePrice = new DoubleWritable();
8
9     @Override
10     protected void reduce(Text key, Iterable<DoubleWritable> values, Context context) throws IOException, InterruptedException {
11         double sum = 0;
12         int count = 0;
13
14         for (DoubleWritable value : values) {
15             sum += value.get();
16             count++;
17         }
18
19         if (count > 0) {
20             averagePrice.set(sum / count);
21             context.write(key, averagePrice);
22         }
23     }
24 }
```

Figure 4.2 : AveragePriceReducer.java file code

```

1  import org.apache.hadoop.io.DoubleWritable;
2  import org.apache.hadoop.io.LongWritable;
3  import org.apache.hadoop.io.Text;
4  import org.apache.hadoop.mapreduce.Mapper;
5  import java.io.IOException;
6
7  public class PriceMapper extends Mapper<LongWritable, Text, Text, DoubleWritable> {
8      private boolean isHeader = true;
9      private Text variety = new Text();
10     private DoubleWritable price = new DoubleWritable();
11
12     @Override
13     protected void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException {
14         String line = value.toString();
15         String[] columns = line.split(",", -1);
16
17         if (isHeader) {
18             isHeader = false;
19             return;
20         }
21
22         if (columns.length < 5) return;
23
24         String priceStr = columns[3].trim();
25         String varietyStr = columns[4].trim();
26
27         if (priceStr.isEmpty() || varietyStr.isEmpty()) return;
28
29         try {
30             double priceVal = Double.parseDouble(priceStr);
31             variety.set(varietyStr);
32             price.set(priceVal);
33             context.write(variety, price);
34         } catch (NumberFormatException e) {
35
36         }
37     }
38 }
39
40 }

```

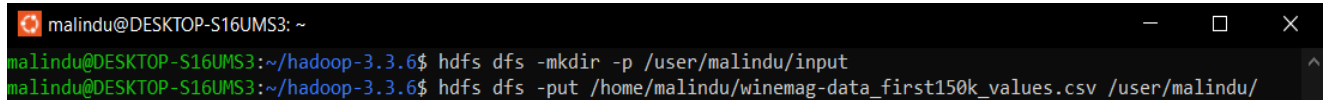
Figure 4.3 : PriceMapper.java file code

## 5 Execution and Testing

This section outlines how to execute and test the MapReduce job in the Hadoop ecosystem. It covers uploading the dataset to HDFS, running the compiled JAR file, and inspecting the output to verify successful execution.

### 5.1 Upload Dataset to HDFS

First, ensure that HDFS is running. Then, create a directory for input if it doesn't already exist and upload the dataset file to that directory.

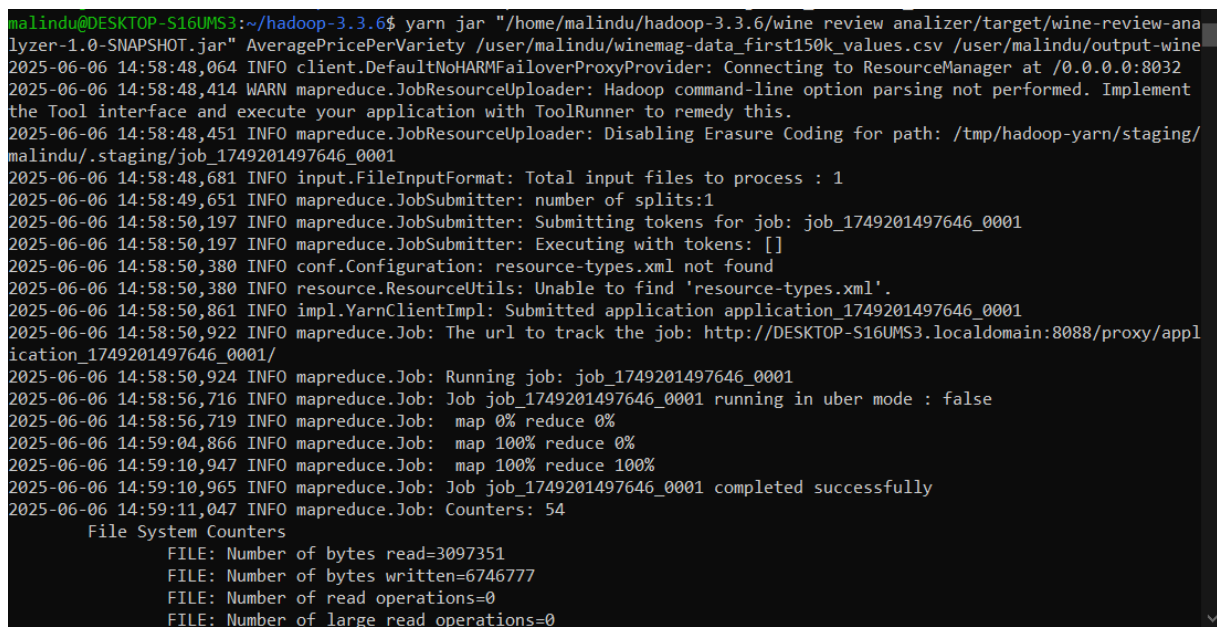


```
malindu@DESKTOP-S16UMS3: ~  
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$ hdfs dfs -mkdir -p /user/malindu/input  
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$ hdfs dfs -put /home/malindu/winemag-data_first150k_values.csv /user/malindu/
```

Figure 5.1 : Creating HDFS directory and uploading a CSV file in Hadoop

### 5.2 Run MapReduce Job

Ensure the Java classes are compiled and the JAR file has been created. We can execute MapReduce job using the following command.



```
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$ yarn jar "/home/malindu/hadoop-3.3.6/wine review analyzer/target/wine-review-analyzer-1.0-SNAPSHOT.jar" AveragePricePerVariety /user/malindu/winemag-data_first150k_values.csv /user/malindu/output-wine  
2025-06-06 14:58:48,064 INFO client.DefaultNoHARMFailoverProxyProvider: Connecting to ResourceManager at /0.0.0.0:8032  
2025-06-06 14:58:48,414 WARN mapreduce.JobResourceUploader: Hadoop command-line option parsing not performed. Implement the Tool interface and execute your application with ToolRunner to remedy this.  
2025-06-06 14:58:48,451 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/malindu/.staging/job_1749201497646_0001  
2025-06-06 14:58:48,681 INFO input.FileInputFormat: Total input files to process : 1  
2025-06-06 14:58:49,651 INFO mapreduce.JobSubmitter: number of splits:1  
2025-06-06 14:58:50,197 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1749201497646_0001  
2025-06-06 14:58:50,197 INFO mapreduce.JobSubmitter: Executing with tokens: []  
2025-06-06 14:58:50,380 INFO conf.Configuration: resource-types.xml not found  
2025-06-06 14:58:50,380 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.  
2025-06-06 14:58:50,861 INFO impl.YarnClientImpl: Submitted application application_1749201497646_0001  
2025-06-06 14:58:50,922 INFO mapreduce.Job: The url to track the job: http://DESKTOP-S16UMS3.localdomain:8088/proxy/application_1749201497646_0001/  
2025-06-06 14:58:50,924 INFO mapreduce.Job: Running job: job_1749201497646_0001  
2025-06-06 14:58:56,716 INFO mapreduce.Job: Job job_1749201497646_0001 running in uber mode : false  
2025-06-06 14:58:56,719 INFO mapreduce.Job: map 0% reduce 0%  
2025-06-06 14:59:04,866 INFO mapreduce.Job: map 100% reduce 0%  
2025-06-06 14:59:10,947 INFO mapreduce.Job: map 100% reduce 100%  
2025-06-06 14:59:10,965 INFO mapreduce.Job: Job job_1749201497646_0001 completed successfully  
2025-06-06 14:59:11,047 INFO mapreduce.Job: Counters: 54  
File System Counters  
FILE: Number of bytes read=3097351  
FILE: Number of bytes written=6746777  
FILE: Number of read operations=0  
FILE: Number of large read operations=0
```

Figure 5.2 : Hadoop MapReduce job calculating average wine prices per variety

This command runs the AveragePricePerVariety driver class using the specified input and output paths in HDFS.

If the output directory already exists, delete it using following Figure 5.3 showing code.

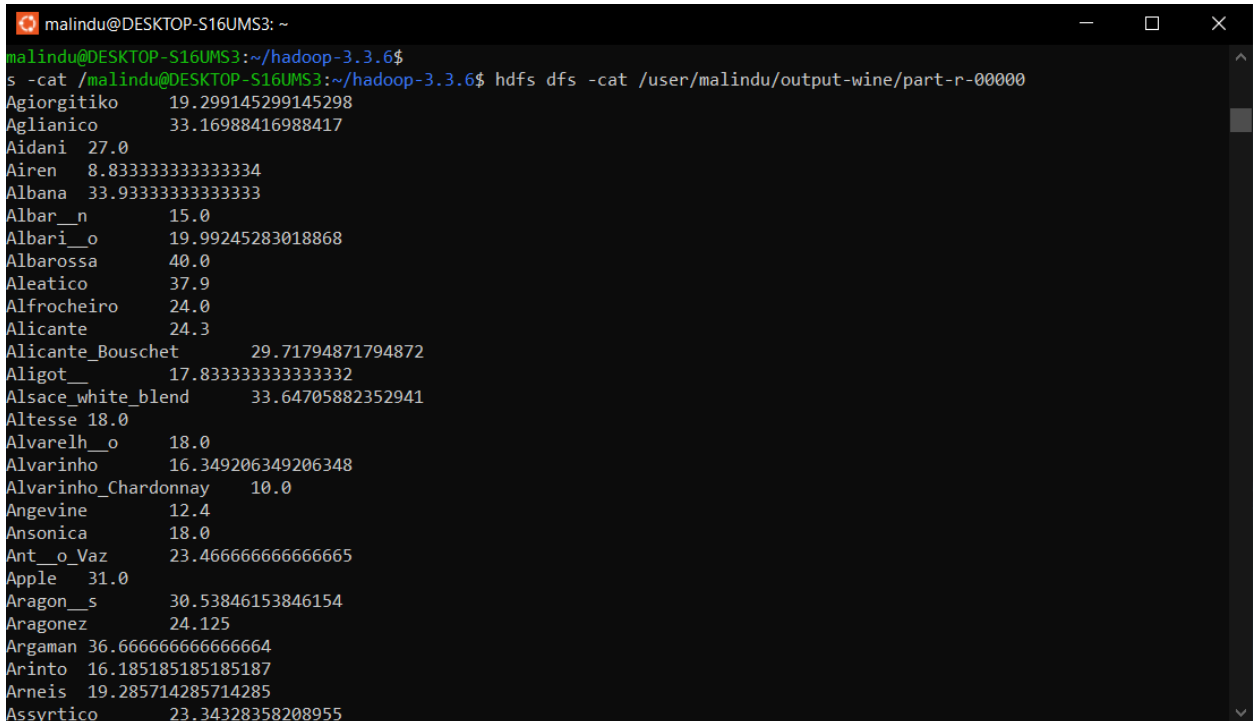
```
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$ hdfs dfs -rm -r /user/malindu/output-wine
```

Figure 5.3 : Command-line interface

Job status can be check via the YARN ResourceManager UI at <http://localhost:8088>.

### 5.3 Output Sample

Once the job finishes, we can read the output stored in the HDFS output directory. It has shown by below Figure 5.4.



```
malindu@DESKTOP-S16UMS3: ~  
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$  
s -cat /malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$ hdfs dfs -cat /user/malindu/output-wine/part-r-00000  
Agiorgitiko 19.299145299145298  
Aglanico 33.16988416988417  
Aidani 27.0  
Airen 8.833333333333334  
Albana 33.93333333333333  
Albar__n 15.0  
Albari__o 19.99245283018868  
Albarossa 40.0  
Aleatico 37.9  
Alfrocheiro 24.0  
Alicante 24.3  
Alicante_Bouschet 29.71794871794872  
Aligot__ 17.833333333333332  
Alsace_white_blend 33.64705882352941  
Altesse 18.0  
Alvarelh__o 18.0  
Alvarinho 16.349206349206348  
Alvarinho_Chardonnay 10.0  
Angevine 12.4  
Ansonica 18.0  
Ant__o_Vaz 23.466666666666665  
Apple 31.0  
Aragon__s 30.53846153846154  
Aragonez 24.125  
Argaman 36.666666666666664  
Arinto 16.185185185185187  
Arneis 19.285714285714285  
Assyrtico 23.34328358208955
```

Figure 5.4 : Hadoop MapReduce output: Average wine prices by variety.

This confirms successful grouping and averaging of wine prices by variety.

## 6 Results and Interpretation

The output consists of wine varieties and their calculated average prices. It demonstrates the ability of Hadoop MapReduce to extract analytical insights from structured but noisy datasets.

```
malindu@DESKTOP-S16UMS3: ~/hadoop-3.3.6
malindu@DESKTOP-S16UMS3:~/hadoop-3.3.6$ hdfs dfs -cat /user/malindu/output-wine/part-r-00000
Agiorgitiko 19.299145299145298
Aglanico 33.16988416988417
Aidani 27.0
Airen 8.833333333333334
Albana 33.93333333333333
Albar_n 15.0
Albar_o 19.99245283018868
Albarossa 40.0
Aleatico 37.9
Alfrocheiro 24.0
Alicante 24.3
Alicante Bouschet 29.71794871794872
Aligot_ 17.833333333333332
Alsace_white_blend 33.64705882352941
Altesse 18.0
Alvarelh_o 18.0
Alvarinho 16.349206349206348
Alvarinho_Chardonnay 18.0
Angevine 12.4
Ansonica 18.0
Ant_o_Vaz 23.466666666666665
Apple 31.0
Aragon_s 30.53846153846154
Aragonez 24.125
Argaman 36.666666666666664
Arinto 16.185185185185187
Arneis 19.285714285714285
Assyrtico 23.34328358208955
Assyrtiko 21.5
Athiri 18.0
Austrian_Red_Blend 37.763636363636365
Austrian_white_blend 28.38888888888889
Auxerrois 24.642857142857142
Avesso 14.666666666666666
Azal 13.0
Baco_Noir 24.22222222222222
Baga 31.625
Baga_Touriga_Nacional 20.0
Barbera 25.901758014477767
Bastardo 30.571428571428573
Bical 15.222222222222221
Black_Monukka 25.0
Black_Muscat 25.923076923076923
Blatina 12.666666666666666
Blauburgunder 19.0
Blauer_Portugieser 15.428571428571429
Blauf_rnkisch 29.026178010471206
Bo_azkere 25.0
Bohal 14.6875
```

Figure 6.1 : Output terminal of wine varieties and their calculated average prices



# 7 Hadoop Web Interfaces

- NameNode: <http://localhost:9870>

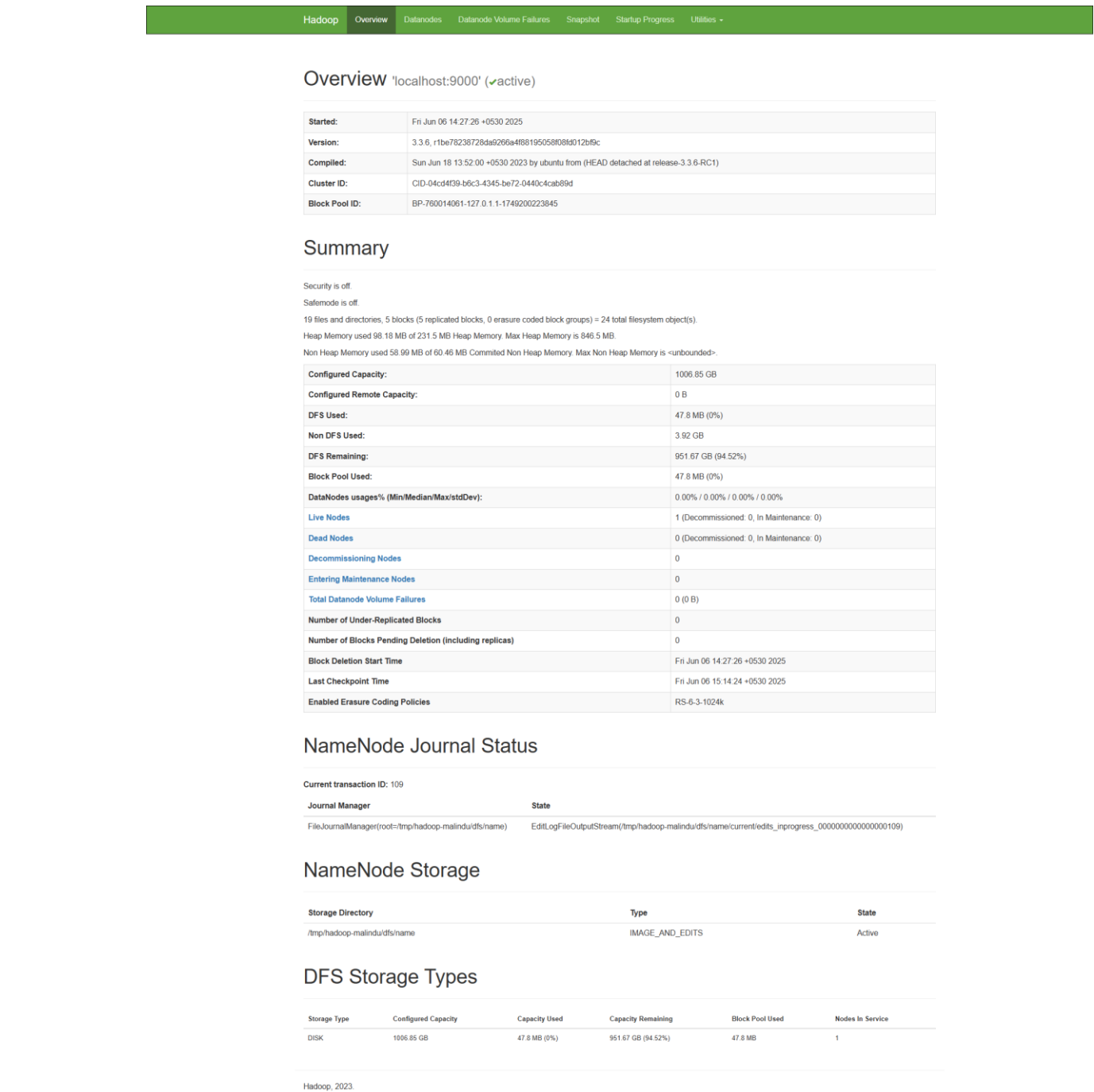


Figure 7.1 : Hadoop web interface (NameNode)

- ResourceManager: <http://localhost:8088>

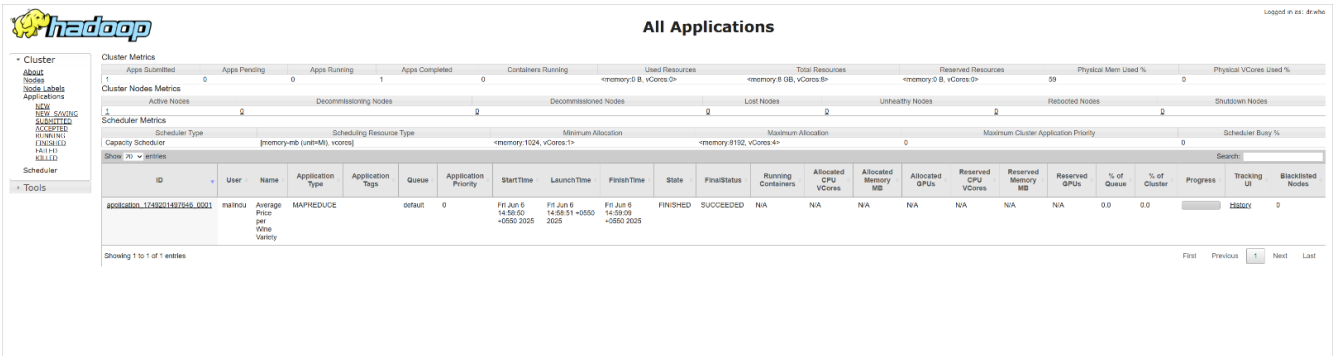


Figure 7.2 : Hadoop web interface (ResourceManager)