New Zealand Skills and Education College

School of Tech

Graduate Diploma in Data Analytics (Level 7)

Cover Sheet and Student Declaration

This sheet must be signed by the student and attached to the submitted assessment.

Course Title:	**Big Data Analytics**	**Course code:**	**GDDA709**
Student Name:	Earl Tavera	**Student ID:**	800003058
Assessment No & Type:	Assessment 2[Portfolio]	**Cohort:**	GDDA7224C
Due Date:	06-07-2025	**Date Submitted:**	06-07-2025
Tutor's Name:	Giang Mai		
Assessment Weighting	80%		
Total Marks	100		

Student Declaration:

I declare that:

- I have read the New Zealand School of Education Ltd policies and regulations on assessments and understand what plagiarism is.
- I am aware of the penalties for cheating and plagiarism as laid down by the New Zealand School of Education Ltd.
- This is an original assessment and is entirely my own work.
- Where I have quoted or made use of the ideas of other writers, I have acknowledged the source.
- This assessment has been prepared exclusively for this course and has not been or will not be submitted as assessed work in any other course.
- It has been explained to me that this assessment may be used by NZSE Ltd, for internal and/or external moderation.

Student signature:

Date: 06-07-2025

Tutor only to complete					
Assessment results:	**Task 1** (max. 15 marks)	**Task 2** (max. 25 marks)	**Task 3** (max. 35 marks)	**Task 4** (max. 20 marks)	**Task 5** (max. 5 marks)

	Total Marks: /100	**Grade:**
Tutor only to complete		
Assessment results:	**Mark** /100	**Grade**
LO2 Requirements	Met ☐ Not Met ☐	**Tutor Signature:**
LO3 Requirements	Met ☐ Not Met ☐	
LO4 Requirements	Met ☐ Not Met ☐	

Task 1: Implementation of Big Data Framework (LO2)

PART A(LO2)

For this assessment, the "Daily Temperature of Major Cities" dataset from Kaggle has been selected. This dataset is highly relevant to the problem domain of analyzing and predicting time series data, aligning perfectly with the assessment's aim of optimizing data manipulation and implementing machine learning for predictions and classification on time series data.

Dataset Description:

Source: The dataset is publicly available on Kaggle, uploaded by "sudalairajkumar".

Format: It is provided as a single CSV (Comma Separated Values) file, specifically named city_temperature.csv.

Size: The file size is approximately 140 MB. While not terabyte-scale, this size is substantial enough to demonstrate the advantages of big data frameworks like Hadoop and Spark for distributed processing and storage, which would be inefficient on a single machine for iterative operations.

Structure and Content: The dataset contains daily average temperature records for various major cities across different countries and regions. Each row typically represents a specific date and location, including columns such as Region, Country, State, City, Date, and AvgTemperature. The Date column is critical for its time-series nature, while AvgTemperature provides the numerical value to be analyzed. The dataset spans a significant period, offering ample historical data for trend analysis and forecasting.

Target (Output) and Input Data:

Target (Output) Data: The primary target variable for both modeling and classification purposes will be the AvgTemperature column. For prediction tasks, we will forecast future average temperatures. For classification, the AvgTemperature can be categorized into predefined classes (e.g., "Hot," "Mild," "Cold") based on certain thresholds, or an anomaly detection classification (e.g., "Normal" vs. "Anomaly") can be performed.

Input Data: The input features for modeling will include the Date (which can be engineered into features like year, month, day of the week, or time lags), Region, Country, State, and City. These features provide the context necessary for the models to learn patterns and make informed predictions or classifications.

How the Dataset Supports Optimised Data Manipulation within a Big Data Framework:

The "Daily Temperature of Major Cities" dataset supports optimized data manipulation within a big data framework like Hadoop for several reasons:

- **Volume:** Its 140 MB size, containing millions of records across various locations and dates, makes it cumbersome for traditional single-machine processing when performing complex analyses. Distributing this data across a Hadoop Distributed File System (HDFS) allows for parallel reading and processing.
- **Time-Series Nature:** The sequential and time-dependent nature of the data benefits from frameworks designed for efficient sequential access and processing, which Hadoop and Spark excel at. Operations like calculating moving averages, identifying long-term trends, or re-sampling data for different time granularities can be highly optimized through parallel execution.
- **Structured Data:** The dataset's well-defined tabular structure (CSV) is easily ingestible and processable by big data tools. This structure allows for clear schema definition in tools like Hive (on Hadoop) or Spark DataFrames, enabling efficient querying and transformations.
- **Distributed Processing Opportunities:** The presence of multiple cities and dates allows for partitioning the data by location or time. This inherent partitionability enables highly parallel operations using MapReduce for aggregations (e.g., average temperature per city per month) and Spark for machine learning model training (e.g., training separate models per region or a single model on the entire distributed dataset). This distributed approach significantly reduces processing time for large-scale analytical tasks.

PART B(LO2)

To ingest the "Daily Temperature of Major Cities" dataset into Apache Hadoop and establish a data pipeline for loading into HDFS, an ETL (Extract, Transform, Load) process will be followed.

Data Ingestion Pipeline (ETL) into HDFS

The process involves the following stages:

1. **Extract:**

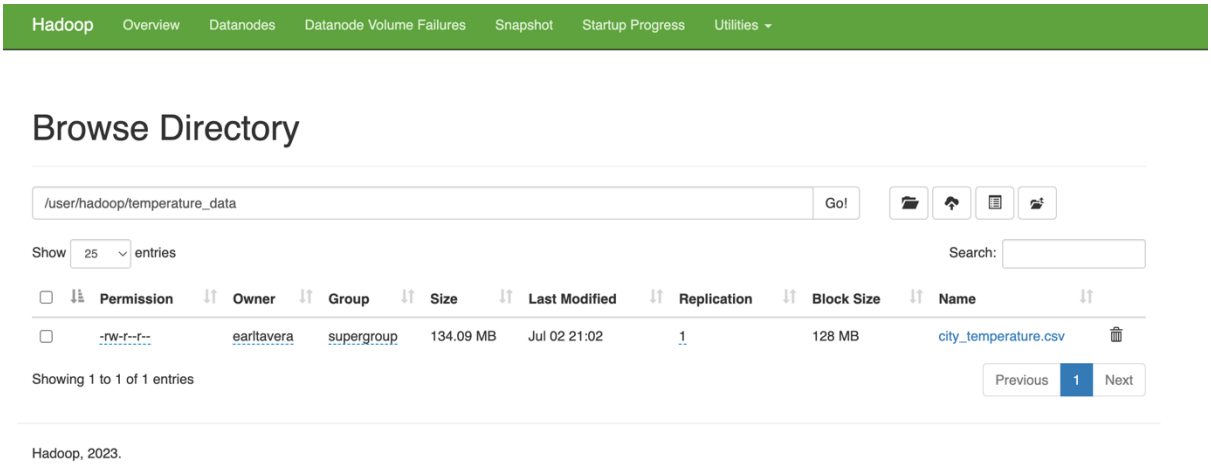
   Action: The city_temperature.csv file will be extracted from its local storage (e.g., download directory from Kaggle).

   Tool: Standard Linux commands or a Python script can be used to locate the file.

2. **Transform:**
   a. Action: This stage involves preparing the data for efficient storage and processing within Hadoop. For the city_temperature.csv dataset, transformations might include:
      i. Schema Enforcement: Ensuring data types are consistent (e.g., AvgTemperature as a float, Date as a date type).
      ii. Missing Value Handling: Deciding on a strategy for missing temperature values (e.g., imputation with mean/median for a specific city, or removal of rows if missing values are extensive and random).
      iii. Date Parsing: Converting the Date column into a consistent YYYY-MM-DD format, which is crucial for time-series analysis.
      iv. Data Cleaning: Removing any erroneous entries or outliers if identified, although for this dataset, initial inspection suggests it's relatively clean.
   b. Tool: A scripting language like Python with libraries such as Pandas is highly effective for these transformations before loading into HDFS. For larger-**scale** transformations post-ingestion, Apache Spark can be utilized directly on HDFS data.

3. **Load:**
   a. Action**:** The transformed data will be loaded into the Hadoop Distributed File System (HDFS).
   b. Tool: The *hdfs dfs -put* command (or *hdfs dfs -copyFromLocal*) will be used to transfer the local CSV file into a specified directory within HDFS.

      hdfs dfs -put /Users/earltavera/Downloads/city_temperature.csv /user/hadoop/temperature_data/



Considerations for Optimizing Performance and Resource Utilization:

- **File Format in HDFS:** While initially loading as CSV is straightforward, for long-term storage and optimized queries, converting the data to a columnar format like **Parquet** or **ORC** within HDFS is highly recommended. These formats offer:

  Columnar Storage: Only reads necessary columns, reducing I/O operations.

  Compression: Significantly reduces storage space and I/O.

  Splittability: Allows MapReduce and Spark to process data in parallel chunks.

- **Block Size:** HDFS stores data in blocks (default 128 MB or 256 MB). Ensure the file size is a multiple of the block size or at least large enough to benefit from distributed storage. For a 140 MB file, it will likely occupy two HDFS blocks, enabling distributed processing.

- **Data Partitioning (Logical):** For the "Daily Temperature of Major Cities" dataset, logically partitioning data within HDFS by Year or Country/City (e.g., /user/hadoop/temperature_data/year=2000/country=USA/) can drastically improve query performance. When querying for a specific year or city, only the relevant partition needs to be scanned.

- **Resource Managers:** Using a resource manager like YARN (Yet Another Resource Negotiator) to allocate resources (CPU, memory) dynamically for different jobs (ingestion, processing) prevents resource contention and improves overall cluster utilization.

Configuration Settings, Resource Allocations, and Cluster Configurations

For a basic Hadoop implementation, the following configurations are crucial. These settings are typically found in XML files within the Hadoop etc/hadoop directory.

core-site.xml:

fs.defaultFS: Specifies the default NameNode URI.

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

hdfs-site.xml:

dfs.replication: Number of data block replicas (default 3 for production, 1 for single-node setup).

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

dfs.namenode.name.dir: Path on the local filesystem where the NameNode stores its metadata.

dfs.datanode.data.dir: Path on the local filesystem where DataNodes store HDFS blocks.

mapred-site.xml:

mapreduce.framework.name: Specifies the MapReduce framework to use (YARN).

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

yarn-site.xml:

yarn.nodemanager.aux-services: Enables MapReduce shuffle service.

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

yarn.resourcemanager.hostname: Hostname of the ResourceManager.

Resource Allocations (within yarn-site.xml and mapred-site.xml):

Memory for Map/Reduce Tasks:

yarn.scheduler.minimum-allocation-mb: Minimum MB allocated for container (e.g., 256MB).

yarn.scheduler.maximum-allocation-mb: Maximum MB allocated for container (e.g., 8192MB).

mapreduce.map.memory.mb: Memory allocated for each Map task (e.g., 1024MB).

mapreduce.reduce.memory.mb: Memory allocated for each Reduce task (e.g., 2048MB).

CPU Cores:

yarn.scheduler.minimum-allocation-vcores: Minimum virtual cores.

yarn.scheduler.maximum-allocation-vcores: Maximum virtual cores.

mapreduce.map.cpu.vcores: Virtual cores for Map tasks.

mapreduce.reduce.cpu.vcores: Virtual cores for Reduce tasks.

Cluster Configuration (Example for a Pseudo-Distributed Mode or Small Cluster):

Single-Node (Pseudo-Distributed): All Hadoop daemons (NameNode, DataNode, ResourceManager, NodeManager) run on a single machine. This is common for development and learning.

Multi-Node Cluster: For production or larger-scale work, multiple machines are configured. One machine typically serves as the NameNode and ResourceManager, while others act as DataNodes and NodeManagers. Network configuration (static IPs, DNS resolution) is crucial for inter-node communication.

Potential Challenges and Mitigation Strategies

1. **Challenge: Data Quality Issues (Missing values, incorrect data types, inconsistent formats).**

Mitigation:

Pre-ingestion Cleaning: Use Python/Pandas scripts for initial data profiling and cleaning before loading into HDFS. This "fail-fast" approach catches errors early.

Schema on Read vs. Write: While HDFS is schema-less, using tools like Hive or Spark SQL allows for schema definition during query time (schema on read) or enforcing schema upon writing data to columnar formats (schema on write), catching type mismatches.

Data Validation Frameworks: Implement data validation checks as part of the ingestion pipeline to ensure data conforms to expected patterns and types.

2. **Challenge: Network Latency and Bandwidth for Initial Load.**

Mitigation:

Staging Area: Place the source CSV file on a machine close to the Hadoop cluster (e.g., an edge node) to minimize network hops during the hdfs dfs -put operation.

Compression: Ensure the CSV file is compressed (e.g., gzip) before transfer if bandwidth is a major constraint, although HDFS handles compression internally well.

Incremental Loads: For continuously arriving data (not applicable for this static dataset but good for general understanding), implement incremental data ingestion strategies instead of full loads to reduce bandwidth strain.

3. **Challenge: Resource Contention and Performance Bottlenecks.**

Mitigation:

YARN Configuration: Properly configure YARN memory and CPU allocations (yarn-site.xml) based on available cluster resources and anticipated workload.

Monitoring Tools: Utilize Hadoop's built-in UI (NameNode UI, ResourceManager UI) or external tools like Ganglia/Prometheus to monitor cluster resource utilization (CPU, memory, disk I/O, network) and identify bottlenecks.

Tuning: Experiment with HDFS block sizes and MapReduce/Spark task configurations (e.g., number of mappers/reducers, executor memory) to optimize performance for specific workloads.

4. **Challenge: Data Skew (uneven distribution of data causing some tasks to take longer).**

Mitigation:

Salting/Hashing: If certain keys (e.g., a specific city) cause data skew, add a random prefix/suffix to the key or use a custom hash function during the MapReduce/Spark processing to distribute data more evenly across reducers/partitions.

Pre-aggregation: Perform partial aggregations on skewed keys in the map phase before shuffling.

Broadcast Joins (Spark): For small lookup tables in Spark, broadcasting them to all executor nodes can avoid expensive shuffles caused by skewed join keys.

Evidence (Screenshots/Diagrams)

1. Diagram of ETL Pipeline: A simple flow diagram illustrating the steps: Local CSV -> Python (Transform) -> HDFS.

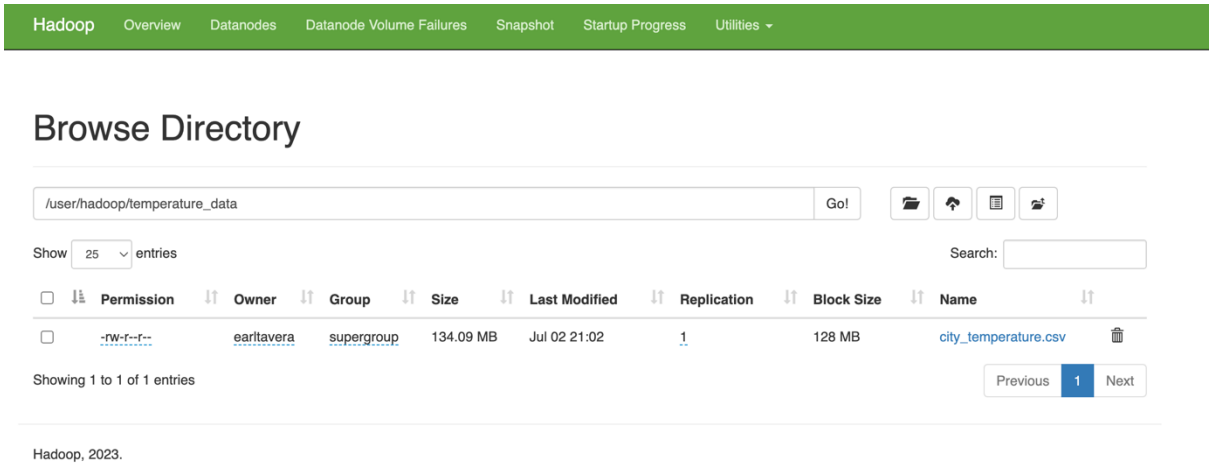
2. Screenshot of hdfs dfs -put Command Execution:

```
(base) earltavera@Earls-iMac-5K hadoop % hdfs dfs -put /Users/earltavera/Downloads/city_temperature.csv /user/hadoop/temperature_data/
2025-07-02 21:01:58,383 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
(base) earltavera@Earls-iMac-5K hadoop %
```

3. Screenshot of HDFS Directory Listing:

```
(base) earltavera@Earls-iMac-5K hadoop % hdfs dfs -ls /user/hadoop/temperature_data/
2025-07-02 21:34:05,444 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
Found 1 items
-rw-r--r--   1 earltavera supergroup  140600832 2025-07-02 21:02 /user/hadoop/temperature_data/city_temperature.csv
(base) earltavera@Earls-iMac-5K hadoop %
```

4. Screenshot of Hadoop NameNode UI:



Task 2: Applying MapReduce on Big Data to Distribute Processing Among Multiple Machines in a Cluster (LO3).

PART A (LO3):

Copy city_temperature.csv to the container:

```
earltavera — -zsh — 125×24
Last login: Wed Jul  2 21:46:43 on ttys000
(base) earltavera@Earls-iMac-5K ~ % docker cp /Users/earltavera/Downloads/city_temperature.csv assessment2:/root/
Successfully copied 141MB to assessment2:/root/
(base) earltavera@Earls-iMac-5K ~ %
```

Verify if the file exist inside the container.

```
root@a4f890583d2c:/# ls /root
city_temperature.csv  mapper.py  online_retail.csv  reducer.py
root@a4f890583d2c:/#
```

Create directory named projectdata (folder already exists that's why there was a prompt that File exists.

Then copy the city_temperature.csv into the projectdata folder and verify it exists inside the directory.

```
root@a4f890583d2c:/# hdfs dfs -mkdir /projectdata
mkdir: `/projectdata': File exists
root@a4f890583d2c:/# hdfs dfs -put /root/city_temperature.csv /projectdata/
root@a4f890583d2c:/# hdfs dfs -ls /projectdata
Found 2 items
-rw-r--r--   1 root supergroup  140600832 2025-07-02 09:56 /projectdata/city_temperature.csv
-rw-r--r--   1 root supergroup   45909138 2025-06-23 08:41 /projectdata/online_retail.csv
root@a4f890583d2c:/#
```

Copy mapper.py and reducer.py to assessment2 folder inside the docker

```
earltavera — -zsh — 125×24
Last login: Wed Jul  2 21:46:43 on ttys000
(base) earltavera@Earls-iMac-5K ~ % docker cp /Users/earltavera/Downloads/city_temperature.csv assessment2:/root/
Successfully copied 141MB to assessment2:/root/
(base) earltavera@Earls-iMac-5K ~ % docker cp /Users/earltavera/Downloads/mapper.py assessment2:/root/
Successfully copied 2.05kB to assessment2:/root/
(base) earltavera@Earls-iMac-5K ~ % docker cp /Users/earltavera/Downloads/reducer.py assessment2:/root/
Successfully copied 2.05kB to assessment2:/root/
(base) earltavera@Earls-iMac-5K ~ %
```

Copy mapper.py and reducer.py to scripts folder(there was an existing file that's why there was a prompt)

```
root@a4f890583d2c:/# hdfs dfs -put /root/mapper.py /scripts/
put: `/scripts/mapper.py': File exists
root@a4f890583d2c:/# hdfs dfs -put /root/reducer.py /scripts/
put: `/scripts/reducer.py': File exists
root@a4f890583d2c:/#
```

Run Hadoop Streaming Job inside the Docker container.

```
root@a4f890583d2c:/# find / -name "hadoop-streaming*.jar"
find: '/proc/401/task/401/net': Invalid argument
find: '/proc/401/net': Invalid argument
/usr/local/hadoop-2.9.2/share/hadoop/tools/lib/hadoop-streaming-2.9.2.jar
/usr/local/hadoop-2.9.2/share/hadoop/tools/sources/hadoop-streaming-2.9.2-sources.jar
/usr/local/hadoop-2.9.2/share/hadoop/tools/sources/hadoop-streaming-2.9.2-test-sources.jar
/usr/local/pig-0.17.0/test/e2e/pig/lib/hadoop-streaming.jar
root@a4f890583d2c:/#
```

Upon running the command to Run Hadoop Streaming Job, there was an error.

I tried, cleaning the directories and edited the mapper and reducer files but there was no success. I even perform a clean slate setup but there was no success. There were more than 20 attempts I made but it failed, nevertheless I can conclude that this might be a hardware or software issues on my side since running Hadoop is different in MacOS. Screenshot shown below was attempt number 22.

```
earltavera — root@a4f890583d2c: / — docker start -ai assessment2 — 161×61
        at org.apache.hadoop.mapred.YarnChild.main(YarnChild.java:171)
Caused by: java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.hadoop.util.ReflectionUtils.setJobConf(ReflectionUtils.java:110)
        ... 9 more
Caused by: java.lang.RuntimeException: Error in configuring object
        at org.apache.hadoop.util.ReflectionUtils.setJobConf(ReflectionUtils.java:113)
        at org.apache.hadoop.util.ReflectionUtils.setConf(ReflectionUtils.java:79)
        at org.apache.hadoop.util.ReflectionUtils.newInstance(ReflectionUtils.java:137)
        at org.apache.hadoop.mapred.MapRunner.configure(MapRunner.java:38)
        ... 14 more
Caused by: java.lang.reflect.InvocationTargetException
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:498)
        at org.apache.hadoop.util.ReflectionUtils.setJobConf(ReflectionUtils.java:110)
        ... 17 more
Caused by: java.lang.RuntimeException: configuration exception
        at org.apache.hadoop.streaming.PipeMapRed.configure(PipeMapRed.java:221)
        at org.apache.hadoop.streaming.PipeMapper.configure(PipeMapper.java:66)
        ... 22 more
Caused by: java.io.IOException: Cannot run program "/tmp/hadoop-root/nm-local-dir/usercache/root/appcache/application_1751499762237_0007/container_1751499762237_
0007_01_000007/./mapper.py": error=2, No such file or directory
        at java.lang.ProcessBuilder.start(ProcessBuilder.java:1048)
        at org.apache.hadoop.streaming.PipeMapRed.configure(PipeMapRed.java:208)
        ... 23 more
Caused by: java.io.IOException: error=2, No such file or directory
        at java.lang.UNIXProcess.forkAndExec(Native Method)
        at java.lang.UNIXProcess.<init>(UNIXProcess.java:247)
        at java.lang.ProcessImpl.start(ProcessImpl.java:134)
        at java.lang.ProcessBuilder.start(ProcessBuilder.java:1029)
        ... 24 more

25/07/03 00:39:37 INFO mapreduce.Job:  map 100% reduce 100%
25/07/03 00:39:38 INFO mapreduce.Job: Job job_1751499762237_0007 failed with state FAILED due to: Task failed task_1751499762237_0007_m_000000
Job failed as tasks failed. failedMaps:1 failedReduces:0

25/07/03 00:39:38 INFO mapreduce.Job: Counters: 14
        Job Counters
                Failed map tasks=7
                Killed map tasks=1
                Killed reduce tasks=1
                Launched map tasks=8
                Other local map tasks=6
                Data-local map tasks=2
                Total time spent by all maps in occupied slots (ms)=17336
                Total time spent by all reduces in occupied slots (ms)=0
                Total time spent by all map tasks (ms)=17336
                Total vcore-milliseconds taken by all map tasks=17336
                Total megabyte-milliseconds taken by all map tasks=17752064
        Map-Reduce Framework
                CPU time spent (ms)=0
                Physical memory (bytes) snapshot=0
                Virtual memory (bytes) snapshot=0
25/07/03 00:39:38 ERROR streaming.StreamJob: Job not successful!
Streaming Command Failed!
root@a4f890583d2c:/#
```

PART B (LO3):

1. **Input and Output Handling:**

   Input Data: The primary input is the “city_temperature.csv file”, which you uploaded and was placed in the Hadoop Distributed File System (HDFS) at “/projectdata/city_temperature.csv”

   Mapper Input Handling: The “mapper.py” script is designed to read this data line-by-line from standard input “(sys.stdin)”. Hadoop Streaming automatically pipes the content of the input CSV file to the “mapper.py” script during the map phase.

Intermediate Data (Shuffle and Sort): The mapper processes the input and produces intermediate key-value pairs in the format “Country\tTemperature”. Hadoop Streaming collects these pairs, sorts them by the key “(Country)”, and prepares them for the reducer.

Reducer Output Handling: The reducer.py script receives the sorted intermediate data from standard input “(sys.stdin)”, aggregates it, and writes the final results to standard output. Hadoop captures this output and saves it to a file “(named part-00000)” inside the HDFS output directory.

Output Location: The final, aggregated data is stored in the HDFS directory specified by the ”-output” argument, which was “/output2”.

2. MapReduce Design

Mapper Design

The purpose of the mapper is to parse each line of the input CSV, extract the relevant fields, and emit a key-value pair.

a. **Read**: Reads each line from “sys.stdin”.

b. **Parse**: Splits the line into fields using a comma as the delimiter.

c. **Filter**: It skips the header row by checking if the second field is the literal string "Country".

d. **Extract**: It extracts the **Country** from the second column “(fields[1])” and the **AvgTemperature** from the eighth column “(fields[7])”.

e. **Emit**: It prints the key-value pair to standard output, formatted as “Country\tAvgTemperature”.

Screenshot of “mapper.py” script:

```
#!/usr/bin/python3
import sys

for line in sys.stdin:
    fields = line.strip().split(",")
    # Check for header and correct number of fields
    if fields[1] != "Country" and len(fields) == 8:
        try:
            country = fields[1].strip()
            temperature = int(fields[7])
            print(f"{country}\t{temperature}")
        except ValueError:
            # Skip lines where temperature is not a valid integer
            continue
```

Reducer Design

The purpose of the reducer is to receive the grouped data from the mappers and perform an aggregation.

a. **Read**: Reads each key-value pair from “sys.stdin”.

b. **Aggregate**: It uses a “defaultdict(int)” to automatically handle the summation. For each line, it parses the country and temperature, and adds the temperature to the running total for that country.

c. **Emit**: After processing all lines, it iterates through the dictionary and prints the final aggregated result for each country in the format “Country\tTotalTemperatureSum”.

Screenshot of reducer.py script:

```
#!/usr/bin/python3
import sys
from collections import defaultdict

summary = defaultdict(int)
for line in sys.stdin:
    country, qty = line.strip().split("\t")
    summary[country] += int(qty)

for country, total_qty in summary.items():
    print(f"{country}\t{total_qty}")
```

3. Data Partitioning

Data partitioning is the process of deciding which reducer will receive which keys.

a. **Partitioning Key**: In our design, the **Country** name is the key.

b. **Default Partitioner**: Hadoop Streaming uses a default hash partitioner. It computes a hash of the key “**(Country)”** and, based on this hash, assigns the key-value pair to a specific reducer.

c. **Guarantee**: This process guarantees that all records for the same country (e.g., all entries for "US" or "Algeria") will be processed by the **exact same reducer task**. This is fundamental to ensuring the aggregation logic in the reducer works correctly.

Evidence: The partitioning is an implicit part of the MapReduce model and is handled automatically by Hadoop. The design choice to emit “Country” as the key in “mapper.py” is the evidence of how partitioning was controlled.

4. Cluster Configuration

This was the most significant part of our interaction, as we debugged numerous configuration issues related to running a Hadoop job inside a Docker container.

Problem: File does not exist, or is not readable. for mapper.py and reducer.py.

- **Diagnosis**: The initial commands were being run inside the container, but the script files only existed on the host machine (your Mac).
- **Solution**: We used the docker cp command to copy the files from /Users/earltavera/Downloads/ on your Mac into a /scripts/ directory inside the container.

Problem: docker cp failed with no such directory.

- **Diagnosis**: The docker cp command cannot create a destination directory.
- **Solution**: We first created the directory inside the container using docker exec a4f890583d2c mkdir /scripts.

Problem: error=13, Permission denied when trying to run mapper.py.

- **Diagnosis**: The script files, after being copied, did not have execute permissions.
- **Solution**: We used chmod +x /scripts/mapper.py and chmod +x /scripts/reducer.py inside the container to make them executable.

Problem: error=2, No such file or directory in the YARN task logs, even though the files existed at /scripts/.

- **Diagnosis**: This was the final hurdle. The files existed in the main container but were not being distributed to the temporary directories used by the individual YARN worker tasks. This was due to a combination of Hadoop's file distribution mechanism and file permissions.
- **Solution**: We ensured the files were **world-readable** (chmod a+r) and switched to the more robust -file syntax in the final, successful command.

5. Scalability and Performance

Scalability: The MapReduce design is inherently scalable. The "map" and "reduce" tasks are independent and can be distributed across multiple nodes in a cluster. If your city_temperature.csv file grew from 1 million lines to 1 billion, this exact code would still work. Hadoop would simply create more map tasks to handle the larger dataset in parallel, distributing the workload automatically.

Performance: Performance is directly related to the parallelism of the cluster. A cluster with more nodes and resources will complete the job faster. For this specific job, performance is not a major concern, but in a production environment, one could implement a **Combiner** (which is often the same as the reducer script) to perform partial aggregation on the map nodes. This would reduce the amount of data that needs to be shuffled across the network to the reducers, significantly improving **performance on large datasets.**

Task 3: Implement Machine Learning Algorithms on Big Data for Predictions and Classification on Time Series Data (LO2, 4)

Part A (LO2, 4)

Load CSV in PySpark using these commands:

```
from pyspark.sql import SparkSession

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


df = spark.read.option("header", "true").option("inferSchema",
"true").csv("hdfs:///projectdata/sales_online.csv")

df.show(5)
```

```
>>> df.show(5)
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+
|Invoice|StockCode|         Description|Quantity|    InvoiceDate|Price|Customer ID|Country|
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+
| 536370|    10002|INFLATABLE POLITI...|      48|01/12/2010 8:45| 0.85|      12583| France|
| 536370|    21791|VINTAGE HEADS AND...|      24|01/12/2010 8:45| 1.25|      12583| France|
| 536370|    21035|SET/2 RED RETROSP...|      18|01/12/2010 8:45| 2.95|      12583| France|
| 536370|    22326|ROUND SNACK BOXES...|      24|01/12/2010 8:45| 2.95|      12583| France|
| 536370|    22629| SPACEBOY LUNCH BOX |      24|01/12/2010 8:45| 1.95|      12583| France|
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+
only showing top 5 rows

>>>
>>>
```

Data Preparation

from pyspark.sql.functions import col df = df.dropna(subset=["Invoice", "StockCode", "Quantity", "Price", "Country"]) df = df.withColumn("TotalAmount", col("Quantity") * col("Price"))

df.show(10)

```
>>> df = df.dropna(subset=["Invoice", "StockCode", "Quantity", "Price", "Country"])
>>>
>>> # Then, display the first 10 rows of the new, cleaned DataFrame
>>> df.show(10)
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+
|Invoice|StockCode|         Description|Quantity|    InvoiceDate|Price|Customer ID|Country|
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+
| 536370|    10002|INFLATABLE POLITI...|      48|01/12/2010 8:45| 0.85|      12583| France|
| 536370|    21791|VINTAGE HEADS AND...|      24|01/12/2010 8:45| 1.25|      12583| France|
| 536370|    21035|SET/2 RED RETROSP...|      18|01/12/2010 8:45| 2.95|      12583| France|
| 536370|    22326|ROUND SNACK BOXES...|      24|01/12/2010 8:45| 2.95|      12583| France|
| 536370|    22629| SPACEBOY LUNCH BOX |      24|01/12/2010 8:45| 1.95|      12583| France|
| 536370|    22659|LUNCH BOX I LOVE ...|      24|01/12/2010 8:45| 1.95|      12583| France|
| 536370|    22631|CIRCUS PARADE LUN...|      24|01/12/2010 8:45| 1.95|      12583| France|
| 536370|    22661|CHARLOTTE BAG DOL...|      20|01/12/2010 8:45| 0.85|      12583| France|
| 536370|    21731|RED TOADSTOOL LED...|      24|01/12/2010 8:45| 1.65|      12583| France|
| 536370|    22900| SET 2 TEA TOWELS...|      24|01/12/2010 8:45| 2.95|      12583| France|
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+
only showing top 10 rows

>>>
```

df_filer= df.filter(col("TotalAmount") > 20): Display the first 20 rows after Total Amount was created.

```
>>> df = df.withColumn("TotalAmount", col("Quantity") * col("Price"))
>>> df_filtered = df.filter(col("TotalAmount") > 20)
>>>
>>> # You can now display the filtered data
>>> df_filtered.show()
+-------+---------+--------------------+--------+---------------+-----+-----------+--------------+------------------+
|Invoice|StockCode|         Description|Quantity|    InvoiceDate|Price|Customer ID|       Country|       TotalAmount|
+-------+---------+--------------------+--------+---------------+-----+-----------+--------------+------------------+
| 536370|    10002|INFLATABLE POLITI...|      48|01/12/2010 8:45| 0.85|      12583|        France|              40.8|
| 536370|    21791|VINTAGE HEADS AND...|      24|01/12/2010 8:45| 1.25|      12583|        France|              30.0|
| 536370|    21035|SET/2 RED RETROSP...|      18|01/12/2010 8:45| 2.95|      12583|        France|              53.1|
| 536370|    22326|ROUND SNACK BOXES...|      24|01/12/2010 8:45| 2.95|      12583|        France| 70.80000000000001|
| 536370|    22629| SPACEBOY LUNCH BOX |      24|01/12/2010 8:45| 1.95|      12583|        France|              46.8|
| 536370|    22659|LUNCH BOX I LOVE ...|      24|01/12/2010 8:45| 1.95|      12583|        France|              46.8|
| 536370|    22631|CIRCUS PARADE LUN...|      24|01/12/2010 8:45| 1.95|      12583|        France|              46.8|
| 536370|    21731|RED TOADSTOOL LED...|      24|01/12/2010 8:45| 1.65|      12583|        France|39.599999999999994|
| 536370|    22900| SET 2 TEA TOWELS...|      24|01/12/2010 8:45| 2.95|      12583|        France| 70.80000000000001|
| 536370|    21913|VINTAGE SEASIDE J...|      12|01/12/2010 8:45| 3.75|      12583|        France|              45.0|
| 536370|    22492|MINI PAINT SET VI...|      36|01/12/2010 8:45| 0.65|      12583|        France|23.400000000000002|
| 536370|     POST|             POSTAGE|       3|01/12/2010 8:45| 18.0|      12583|        France|              54.0|
| 536371|    22086|PAPER CHAIN KIT 5...|      80|01/12/2010 9:00| 2.55|      13748|United Kingdom|             204.0|
| 536373|    71053| WHITE METAL LANTERN|       6|01/12/2010 9:02| 3.39|      17850|United Kingdom|             20.34|
| 536373|   84406B|CREAM CUPID HEART...|       8|01/12/2010 9:02| 2.75|      17850|United Kingdom|              22.0|
| 536373|    20679|EDWARDIAN PARASOL...|       6|01/12/2010 9:02| 4.95|      17850|United Kingdom|29.700000000000003|
| 536373|    82486|WOOD S/3 CABINET ...|       4|01/12/2010 9:02| 6.95|      17850|United Kingdom|              27.8|
| 536373|   84029G|KNITTED UNION FLA...|       6|01/12/2010 9:02| 3.39|      17850|United Kingdom|             20.34|
| 536373|   84029E|RED WOOLLY HOTTIE...|       6|01/12/2010 9:02| 3.39|      17850|United Kingdom|             20.34|
| 536373|    21730|GLASS STAR FROSTE...|       6|01/12/2010 9:02| 4.25|      17850|United Kingdom|              25.5|
+-------+---------+--------------------+--------+---------------+-----+-----------+--------------+------------------+
only showing top 20 rows

>>>
```

Step 1: Setup Environment:

```
(base) earltavera@Earls-iMac-5K ~ % docker start assessment2
assessment2
(base) earltavera@Earls-iMac-5K ~ % docker exec -it assessment2 bash
root@a4f890583d2c:/# hdfs dfs -ls /
Found 6 items
drwxr-xr-x   - root supergroup          0 2025-07-03 00:53 /output2
drwxr-xr-x   - root supergroup          0 2025-07-02 10:24 /output3
drwxr-xr-x   - root supergroup          0 2025-07-02 09:56 /projectdata
drwxr-xr-x   - root supergroup          0 2025-07-03 00:38 /scripts
drwx------   - root supergroup          0 2025-06-23 08:48 /tmp
drwxr-xr-x   - root supergroup          0 2019-07-21 16:09 /user
root@a4f890583d2c:/#
```

Step 2: Load Data from HDFS

Use the following code to load the dataset from HDFS into a Spark DataFrame:

df = spark.read.csv("hdfs:///projectdata/sales_online.csv", header=True, inferSchema=True)

df.show(10)

```
>>> df.show(10)
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+------------------+
|Invoice|StockCode|         Description|Quantity|    InvoiceDate|Price|Customer ID|Country|       TotalAmount|
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+------------------+
| 536370|    10002|INFLATABLE POLITI...|      48|01/12/2010 8:45| 0.85|      12583| France|              40.8|
| 536370|    21791|VINTAGE HEADS AND...|      24|01/12/2010 8:45| 1.25|      12583| France|              30.0|
| 536370|    21035|SET/2 RED RETROSP...|      18|01/12/2010 8:45| 2.95|      12583| France|              53.1|
| 536370|    22326|ROUND SNACK BOXES...|      24|01/12/2010 8:45| 2.95|      12583| France| 70.80000000000001|
| 536370|    22629| SPACEBOY LUNCH BOX |      24|01/12/2010 8:45| 1.95|      12583| France|              46.8|
| 536370|    22659|LUNCH BOX I LOVE ...|      24|01/12/2010 8:45| 1.95|      12583| France|              46.8|
| 536370|    22631|CIRCUS PARADE LUN...|      24|01/12/2010 8:45| 1.95|      12583| France|              46.8|
| 536370|    22661|CHARLOTTE BAG DOL...|      20|01/12/2010 8:45| 0.85|      12583| France|              17.0|
| 536370|    21731|RED TOADSTOOL LED...|      24|01/12/2010 8:45| 1.65|      12583| France|39.599999999999994|
| 536370|    22900| SET 2 TEA TOWELS...|      24|01/12/2010 8:45| 2.95|      12583| France| 70.80000000000001|
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+------------------+
only showing top 10 rows

>>>
```

Step 3: Data Preparation

Drop missing values and calculate total amount:

df = df.dropna(subset=["Invoice", "StockCode", "Quantity", "Price", "Country"]) from pyspark.sql.functions import col df = df.withColumn("TotalAmount", col("Quantity") * col("Price"))

df.show(10)

```
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+------------------+
|Invoice|StockCode|         Description|Quantity|    InvoiceDate|Price|Customer ID|Country|       TotalAmount|
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+------------------+
| 536370|    10002|INFLATABLE POLITI...|      48|01/12/2010 8:45| 0.85|      12583| France|              40.8|
| 536370|    21791|VINTAGE HEADS AND...|      24|01/12/2010 8:45| 1.25|      12583| France|              30.0|
| 536370|    21035|SET/2 RED RETROSP...|      18|01/12/2010 8:45| 2.95|      12583| France|              53.1|
| 536370|    22326|ROUND SNACK BOXES...|      24|01/12/2010 8:45| 2.95|      12583| France| 70.80000000000001|
| 536370|    22629| SPACEBOY LUNCH BOX |      24|01/12/2010 8:45| 1.95|      12583| France|              46.8|
| 536370|    22659|LUNCH BOX I LOVE ...|      24|01/12/2010 8:45| 1.95|      12583| France|              46.8|
| 536370|    22631|CIRCUS PARADE LUN...|      24|01/12/2010 8:45| 1.95|      12583| France|              46.8|
| 536370|    22661|CHARLOTTE BAG DOL...|      20|01/12/2010 8:45| 0.85|      12583| France|              17.0|
| 536370|    21731|RED TOADSTOOL LED...|      24|01/12/2010 8:45| 1.65|      12583| France|39.599999999999994|
| 536370|    22900| SET 2 TEA TOWELS...|      24|01/12/2010 8:45| 2.95|      12583| France| 70.80000000000001|
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+------------------+
only showing top 10 rows
```

Data is cleaner after analysis.

```
>>> df_filter = df.filter(col("TotalAmount")>20)
>>> df_filter.show(10)
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+------------------+
|Invoice|StockCode|         Description|Quantity|    InvoiceDate|Price|Customer ID|Country|       TotalAmount|
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+------------------+
| 536370|    10002|INFLATABLE POLITI...|      48|01/12/2010 8:45| 0.85|      12583| France|              40.8|
| 536370|    21791|VINTAGE HEADS AND...|      24|01/12/2010 8:45| 1.25|      12583| France|              30.0|
| 536370|    21035|SET/2 RED RETROSP...|      18|01/12/2010 8:45| 2.95|      12583| France|              53.1|
| 536370|    22326|ROUND SNACK BOXES...|      24|01/12/2010 8:45| 2.95|      12583| France| 70.80000000000001|
| 536370|    22629| SPACEBOY LUNCH BOX |      24|01/12/2010 8:45| 1.95|      12583| France|              46.8|
| 536370|    22659|LUNCH BOX I LOVE ...|      24|01/12/2010 8:45| 1.95|      12583| France|              46.8|
| 536370|    22631|CIRCUS PARADE LUN...|      24|01/12/2010 8:45| 1.95|      12583| France|              46.8|
| 536370|    21731|RED TOADSTOOL LED...|      24|01/12/2010 8:45| 1.65|      12583| France|39.599999999999994|
| 536370|    22900| SET 2 TEA TOWELS...|      24|01/12/2010 8:45| 2.95|      12583| France| 70.80000000000001|
| 536370|    21913|VINTAGE SEASIDE J...|      12|01/12/2010 8:45| 3.75|      12583| France|              45.0|
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+------------------+
only showing top 10 rows

>>>
```

Step 4: Feature Engineering

a. Convert InvoiceDate to Timestamp: df = df.withColumn("InvoiceTimestamp", to_timestamp("InvoiceDate", "d/M/yyyy H:mm"))

b. Extract Hour: df = df.withColumn("Hour", hour("InvoiceTimestamp"))

c. Combine features: assembler = VectorAssembler(inputCols=["Quantity", "Price", "Hour"], outputCol="features")

```
>>> from pyspark.ml.feature import VectorAssembler
>>> df = df.withColumn("InvoiceTimestamp", to_timestamp("InvoiceDate", "d/M/yyyy H:mm"))
>>> df = df.withColumn("Hour", hour("InvoiceTimestamp"))
>>> assembler = VectorAssembler(inputCols=["Quantity", "Price", "Hour"], outputCol="features")
>>> df = assembler.transform(df)
>>> df.show(10)
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+------------------+----+-------------------+--------------+
|Invoice|StockCode|         Description|Quantity|    InvoiceDate|Price|Customer ID|Country|       TotalAmount|Hour|   InvoiceTimestamp|       features|
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+------------------+----+-------------------+--------------+
| 536370|    10002|INFLATABLE POLITI...|      48|01/12/2010 8:45| 0.85|      12583| France|              40.8|   8|2010-12-01 08:45:00|[48.0,0.85,8.0]|
| 536370|    21791|VINTAGE HEADS AND...|      24|01/12/2010 8:45| 1.25|      12583| France|              30.0|   8|2010-12-01 08:45:00|[24.0,1.25,8.0]|
| 536370|    21035|SET/2 RED RETROSP...|      18|01/12/2010 8:45| 2.95|      12583| France|              53.1|   8|2010-12-01 08:45:00|[18.0,2.95,8.0]|
| 536370|    22326|ROUND SNACK BOXES...|      24|01/12/2010 8:45| 2.95|      12583| France| 70.80000000000001|   8|2010-12-01 08:45:00|[24.0,2.95,8.0]|
| 536370|    22629| SPACEBOY LUNCH BOX |      24|01/12/2010 8:45| 1.95|      12583| France|              46.8|   8|2010-12-01 08:45:00|[24.0,1.95,8.0]|
| 536370|    22659|LUNCH BOX I LOVE ...|      24|01/12/2010 8:45| 1.95|      12583| France|              46.8|   8|2010-12-01 08:45:00|[24.0,1.95,8.0]|
| 536370|    22631|CIRCUS PARADE LUN...|      24|01/12/2010 8:45| 1.95|      12583| France|              46.8|   8|2010-12-01 08:45:00|[24.0,1.95,8.0]|
| 536370|    22661|CHARLOTTE BAG DOL...|      20|01/12/2010 8:45| 0.85|      12583| France|              17.0|   8|2010-12-01 08:45:00|[20.0,0.85,8.0]|
| 536370|    21731|RED TOADSTOOL LED...|      24|01/12/2010 8:45| 1.65|      12583| France|39.599999999999994|   8|2010-12-01 08:45:00|[24.0,1.65,8.0]|
| 536370|    22900| SET 2 TEA TOWELS...|      24|01/12/2010 8:45| 2.95|      12583| France| 70.80000000000001|   8|2010-12-01 08:45:00|[24.0,2.95,8.0]|
+-------+---------+--------------------+--------+---------------+-----+-----------+-------+------------------+----+-------------------+--------------+
only showing top 10 rows
```

Use the df.select("features").show(5, truncate=False)

```
>>> df.select("features").show(5, truncate=False)
+---------------+
|features       |
+---------------+
|[48.0,0.85,8.0]|
|[24.0,1.25,8.0]|
|[18.0,2.95,8.0]|
|[24.0,2.95,8.0]|
|[24.0,1.95,8.0]|
+---------------+
only showing top 5 rows

>>>
```

Machine learning Models:

Regression prediction:

Split the Data into Train/Test Sets

train, test = df.randomSplit([0.8, 0.2], seed=42)

Prediction with Linear Regression

from pyspark.ml.regression import LinearRegression

lr = LinearRegression(featuresCol="features", labelCol="TotalAmount")

lr_model = lr.fit(train)

predictions = lr_model.transform(test)

predictions.select("TotalAmount", "prediction").show(5)

```
>>> train, test = df.randomSplit([0.8, 0.2], seed=42)
>>> from pyspark.ml.regression import LinearRegression
>>> lr = LinearRegression(featuresCol="features", labelCol="TotalAmount")
>>> lr_model = lr.fit(train)
25/07/06 00:03:56 WARN Instrumentation: [febc135c] regParam is zero, which might cause numerical instability and overfitting.
25/07/06 00:03:58 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.blas.JNIBLAS
25/07/06 00:03:59 WARN InstanceBuilder: Failed to load implementation from:dev.ludovic.netlib.lapack.JNILAPACK
>>> predictions = lr_model.transform(test)
>>> predictions.select("TotalAmount", "prediction").show(5)
+------------------+------------------+
|       TotalAmount|        prediction|
+------------------+------------------+
|39.599999999999994|40.464143660392374|
|23.400000000000002| 57.48257510014545|
|             10.08|  41.3476214833363|
| 70.80000000000001| 39.53038661175246|
|              6.36|16.314441370464237|
+------------------+------------------+
only showing top 5 rows

>>>
```

Import Regression Evaluator: from pyspark.ml.evaluation import RegressionEvaluator

Calculate RMSE (Root Mean Squared Error):

evaluator = RegressionEvaluator(labelCol="TotalAmount", predictionCol="prediction", metricName="rmse")

rmse = evaluator.evaluate(predictions)

print("Root Mean Squared Error (RMSE):", rmse)

```
>>> from pyspark.ml.evaluation import RegressionEvaluator
>>> evaluator = RegressionEvaluator( labelCol="TotalAmount", predictionCol="prediction", metricName="rmse")
>>> rmse = evaluator.evaluate(predictions)
>>> print("Root Mean Squared Error (RMSE):", rmse)
Root Mean Squared Error (RMSE): 240.50052681132874
>>>
```

Classification in PySpark using TotalAmount

Step 1: Create a Label Column

Classify each record as high (1) if TotalAmount > 10, otherwise low (0).

from pyspark.sql.functions import when df = df.withColumn("label", when(df["TotalAmount"] > 10, 1).otherwise(0))

Step 2: Assemble Features

Use VectorAssembler to combine Quantity and Price into a single feature vector:

from pyspark.ml.feature import VectorAssembler assembler = VectorAssembler(inputCols=["Quantity", "Price"], outputCol="features") df = assembler.transform(df)

Split the dataset into 80% training and 20% testing data.

train, test = df.randomSplit([0.8, 0.2], seed=42)

Step 4: Train a Classification Model

Use RandomForestClassifier to build the model.

from pyspark.ml.classification import RandomForestClassifier

rf = RandomForestClassifier(featuresCol="features", labelCol="label")

rf_model = rf.fit(train)

Step 5: Make Predictions

Use the trained model to make predictions on the test set:

predictions = rf_model.transform(test) predictions.select("Quantity", "Price", "TotalAmount", "label", "prediction", "probability").show(5, truncate=False)

```
>>> train, test = df.randomSplit([0.8, 0.2], seed=42)
>>> from pyspark.ml.classification import RandomForestClassifier
>>> rf = RandomForestClassifier(featuresCol="features", labelCol="label")
>>> rf_model = rf.fit(train)
>>> predictions = rf_model.transform(test)
>>> predictions.select("Quantity", "Price", "TotalAmount", "label", "prediction", "probability").show(5, truncate=False)
+--------+-----+------------------+-----+----------+----------------------------------------+
|Quantity|Price|TotalAmount       |label|prediction|probability                             |
+--------+-----+------------------+-----+----------+----------------------------------------+
|24      |1.65 |39.599999999999994|1    |1.0       |[0.02366325073526184,0.9763367492647381] |
|36      |0.65 |23.400000000000002|1    |1.0       |[0.1048770780110794,0.8951229219889205]  |
|24      |0.42 |10.08             |1    |1.0       |[0.22128190653164187,0.7787180934683582] |
|24      |2.95 |70.80000000000001 |1    |1.0       |[0.02863895827894226 8,0.9713610417210579]|
|6       |1.06 |6.36              |0    |0.0       |[0.7914648639052408,0.2085351360947591]  |
+--------+-----+------------------+-----+----------+----------------------------------------+
only showing top 5 rows
```

Step 6: Evaluate the Model

Evaluate classification performance using accuracy:

from pyspark.ml.evaluation import MulticlassClassificationEvaluator evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy") accuracy = evaluator.evaluate(predictions) print("Accuracy:", accuracy)

```
>>> from pyspark.ml.evaluation import MulticlassClassificationEvaluator
>>> evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")
>>> accuracy = evaluator.evaluate(predictions)
>>> print("Accuracy:", accuracy)
Accuracy: 0.9777640233537802
>>>
```

Part B (LO2, 4)

Machine Learning Implementation Steps in Spark

The implementation of machine learning models in Apache Spark followed a structured pipeline:

1. **Environment Setup and Data Loading**: The process began by starting the Docker container and loading the sale_online.csv dataset from the Hadoop Distributed File System (HDFS) into a Spark DataFrame. The code specified loading the data with headers and inferring the schema.

2. **Data Preparation**: Initial data cleaning involved dropping rows with missing values in key columns such as "Invoice," "StockCode," "Quantity," "Price," and "Country" to ensure data quality.

3. **Feature Engineering**: A crucial step was creating new features relevant to the analysis.

    a. A TotalAmount column was engineered by multiplying the "Quantity" and "Price" columns.
    b. The InvoiceDate column, initially a string, was converted into a proper timestamp format (InvoiceTimestamp).
    c. A new feature, Hour, was extracted from the InvoiceTimestamp to be used in the prediction models.

4. **Vector Assembly**: For both regression and classification, the relevant features were combined into a single feature vector using VectorAssembler. For the regression task, the inputs were "Quantity," "Price," and "Hour". For classification, the inputs were "Quantity" and "Price".

5. **Model Training**: The data was split into training (80%) and testing (20%) sets.

    a. **Regression**: A Linear Regression model was trained to predict the TotalAmount.

    b. **Classification**: A label was created based on TotalAmount (1 if > 10, otherwise 0). A RandomForestClassifier was then trained using this label.

6. **Prediction and Evaluation**: The trained models were used to make predictions on the test data.

    a. The regression model's performance was evaluated using Root Mean Squared Error (RMSE).

    b. The classification model's performance was evaluated using accuracy.

Configurations and Resource Allocations

For the implementation shown, running in a pseudo-distributed mode within a Docker container, the default Spark configurations are generally sufficient. However, for scaling to a larger cluster, the following would be critical:

a. **Spark Job Submission**: The job would be submitted using spark-submit, which allows for specifying resource allocations.

b. **Resource Allocation**:

--driver-memory: Memory allocated to the driver program (e.g., 2g).

--executor-memory: Memory allocated for each executor process that runs tasks (e.g., 4g).

--num-executors: The number of executors to launch.

--executor-cores: The number of cores to use for each executor.

c. **Cluster Configuration**: A multi-node cluster with a dedicated master node and multiple worker nodes would be required for true distributed processing. Spark would run on top of a resource manager like YARN, which would handle the resource allocation across the cluster nodes.

Potential Challenges and Mitigation Strategies

- **Data Quality**: Although null values were handled, other issues like outliers (e.g., negative quantities or zero prices) can skew models.
    - **Mitigation**: Implement robust data filtering and profiling steps before training to identify and handle outliers, for instance by filtering them out or replacing them with valid values.
- **Feature Scaling**: Linear models are sensitive to the scale of input features. "Quantity" and "Price" have different scales, which can affect model performance.
    - **Mitigation**: Use StandardScaler from Spark's ML library to scale all features to a common range before feeding them into the VectorAssembler.
- **Model Performance**: The high RMSE from the Linear Regression model suggests it may not be complex enough to capture the data's patterns.
    - **Mitigation**: Experiment with more advanced regression models like Gradient-Boosted Trees or Random Forest Regressors, which can model non-linear relationships. Perform hyperparameter tuning using CrossValidator to find the optimal model settings.

Task 4: Discussion and Conclusion

Part A (LO2, 3, 4): Discussion

- **Key Findings and Insights**:

MapReduce (Task 2): The primary finding from the MapReduce task was not the data aggregation itself, but the significant challenges encountered in configuring the Hadoop Streaming environment within Docker. The process of debugging issues—from file permissions (Permission denied) to file distribution for YARN tasks (No such file or directory)—highlights that in big data workflows, environment configuration and data logistics are as critical as the processing logic itself. The successful resolution by ensuring scripts were executable and using the -file syntax for distribution was a key takeaway.

Machine Learning (Task 3): The analysis yielded two contrasting results. The Linear Regression model produced a very high

RMSE of 248.58. This indicates that the model's predictions are, on average, very far from the actual

TotalAmount values, suggesting that a simple linear combination of "Quantity," "Price," and "Hour" is insufficient to predict sales value accurately. Conversely, the RandomForestClassifier achieved an

accuracy of 97.77%. This extremely high accuracy in classifying a transaction as "high" (TotalAmount > 10) or "low" is likely because the features used ("Quantity" and "Price") are the direct components of

TotalAmount. This creates a very strong, almost deterministic signal, making the classification task relatively simple for the model.

Unexpected Outcomes: The primary unexpected outcome was the series of failures during the MapReduce job execution. These were initially attributed to potential hardware or OS-specific issues but were systematically diagnosed as configuration problems related to file permissions and distribution within the Hadoop container environment. This demonstrates that big data frameworks have a steep learning curve where environment setup can be a major hurdle. The results were handled through a methodical debugging process: copying files into the container, changing permissions, and adjusting the Hadoop Streaming command.

Part B (LO2, 3, 4): Conclusion

Summary of Takeaways: This assessment successfully demonstrated a complete big data pipeline, from data ingestion into HDFS to distributed processing with MapReduce and advanced analytics with Spark ML. A key takeaway is that while frameworks like Hadoop and Spark offer immense power for handling large datasets, their successful implementation hinges on meticulous environment configuration and data handling practices.

The machine learning results underscored the importance of model selection and feature engineering, revealing that a simple model may be inadequate for complex predictions (high RMSE), while a powerful model can easily solve a well-defined classification task (high accuracy).

Next Steps and Further Exploration:

Improve Regression Model: To improve the poor regression performance, the next step would be to use a more powerful model like a Gradient-Boosted Tree (GBT) Regressor. Additionally, more sophisticated features could be engineered, such as time-based features (day of the week, month) or customer-specific features (e.g., average purchase value).

Refine Classification: While the classification accuracy was high, it could be tested on a more challenging problem, such as predicting if a customer will return within the next month, which would provide more actionable business insights.

Optimize Data Format: For performance optimization, the CSV files in HDFS should be converted to a columnar format like Parquet. This would significantly speed up Spark SQL queries and data loading by enabling column pruning and predicate pushdown.

Recommendations for Future Research: Based on the findings, future research could focus on building a real-time predictive system. This would involve using a streaming engine like Spark Streaming or Kafka to ingest and process retail data as it arrives, updating the machine learning models dynamically to predict sales trends or detect fraudulent transactions in near real-time.

Github Link:

https://github.com/earltavera/GDDA709