COURSEWORK: Academic Year 2025/2026

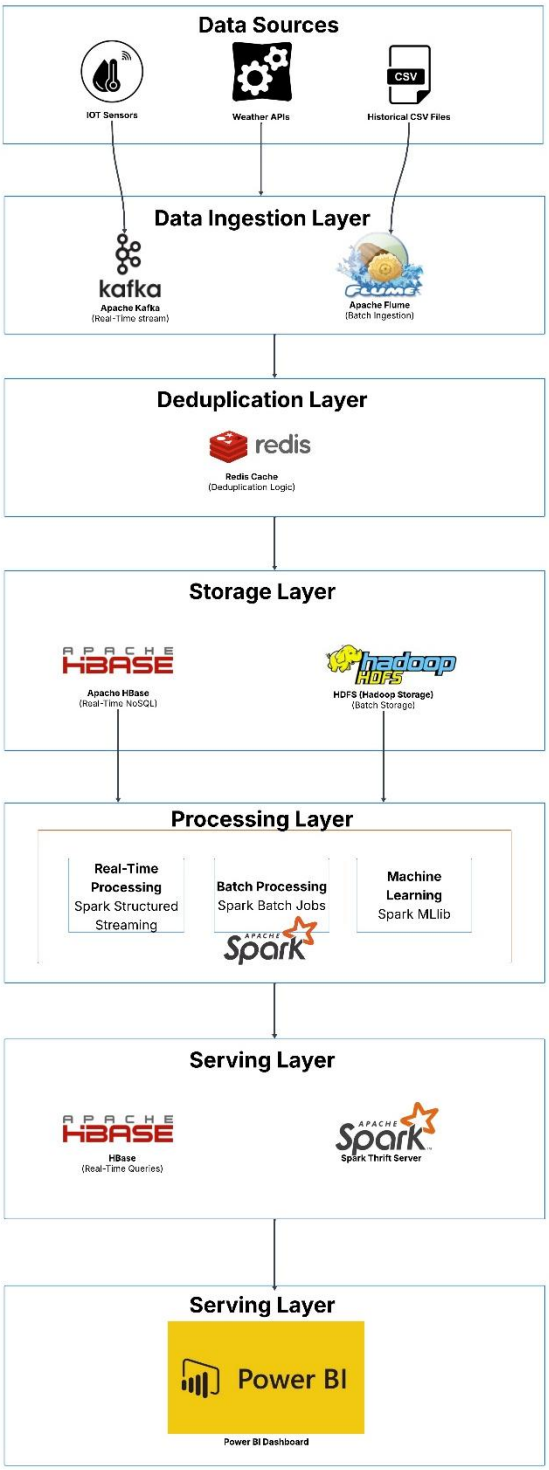
CMM705 Big Data Programming

Anjana Dodampe
RGU ID: 2522702
IIT ID: 20250693

Table of Contents

[Task 1] - Designing a Solution Architecture

[Task 1] [1] Solution Architecture Design



[Task 1] [2] Roles of the components of the Designed Solution Architecture

1. Data Ingestion Layer

Apache Kafka → Collects live weather data

- Weather stations send data every 10-15 minutes
- Stores messages for 7 days (backup if processing fails)
- Partitions data by district (parallel processing)

Apache Flume → Loads historical CSV files

- Monitors folder for new CSV files
- Automatically uploads to HDFS
- Runs hourly batch ingestion

2. Deduplication Layer

Redis Cache → Prevents duplicate records

- Creates unique key: district:timestamp:temperature
- Check if data already seen in last 2 hours
- If duplicate → discard, if new → pass to storage
- Saves 30-40% storage by removing duplicates

3. Storage Layer

HBase → Fast access to recent data (last 30 days)

- Stores latest readings for real-time dashboard
- Query speed: <10ms (instant dashboard updates)
- Auto-deletes old data after 30 days (stays lean)

HDFS → Long-term historical storage (15+ years)

- Keeps all raw and processed data
- Scales to petabytes (add more nodes as needed)
- 3x replication (survives hardware failures)
- Uses Parquet format (10x smaller than CSV)

4. Processing Layer

Apache Spark → Does all the work (one tool for everything)

- Spark Streaming: Processes live Kafka data every 10 seconds
    - Reads messages → deduplicates → writes to HBase + HDFS
- Spark Batch: Scheduled jobs for aggregations
    - Daily: Calculate average temp, total rainfall per district
    - Hourly: Clean data, remove outliers
- Spark MLlib: Machine learning predictions
    - Trains model to predict evapotranspiration (ET_0)
    - Help farmers know how much to irrigate

5. Serving Layer

HBase REST API → Dashboard queries latest data

- When a user through Power BI asks "What's Colombo's current temperature?"
- HBase responds in <10ms with live data

Spark Thrift Server → Historical SQL queries

- When a user through Power BI asks "What was average rainfall in 2023?"
- Spark SQL queries HDFS and returns results
- Analysts run custom SQL queries

6. Visualization Layer

Power BI Dashboard → User interface

- Real-time panel: Shows live weather (refreshes every 30 sec)
- Historical trends: Charts showing patterns over years
- ML predictions: Evapotranspiration forecasts for irrigation planning

[Task 2] Data Analysis

Task 2 [1] 1 – Hadoop MapReduce Analysis - Final Output:

RGU_ID_2522702_IIT_ID_20250693 > Task_2 > task_2-1_mapreduce > outputs > task2-1-1_output.txt > data

```
Ampara had a total precipitation of 2852 hours with a mean temperature of 25 for 1st month
Ampara had a total precipitation of 2476 hours with a mean temperature of 25 for 2nd month
Ampara had a total precipitation of 1871 hours with a mean temperature of 27 for 3rd month
Ampara had a total precipitation of 1689 hours with a mean temperature of 28 for 4th month
Ampara had a total precipitation of 1466 hours with a mean temperature of 28 for 5th month
Ampara had a total precipitation of 693 hours with a mean temperature of 29 for 6th month
Ampara had a total precipitation of 708 hours with a mean temperature of 30 for 7th month
Ampara had a total precipitation of 816 hours with a mean temperature of 29 for 8th month
Ampara had a total precipitation of 1169 hours with a mean temperature of 28 for 9th month
Ampara had a total precipitation of 2523 hours with a mean temperature of 27 for 10th month
Ampara had a total precipitation of 3570 hours with a mean temperature of 26 for 11th month
Ampara had a total precipitation of 3652 hours with a mean temperature of 25 for 12th month
Anuradhapura had a total precipitation of 1461 hours with a mean temperature of 25 for 1st month
Anuradhapura had a total precipitation of 941 hours with a mean temperature of 26 for 2nd month
Anuradhapura had a total precipitation of 793 hours with a mean temperature of 27 for 3rd month
Anuradhapura had a total precipitation of 1360 hours with a mean temperature of 28 for 4th month
Anuradhapura had a total precipitation of 1925 hours with a mean temperature of 28 for 5th month
Anuradhapura had a total precipitation of 1182 hours with a mean temperature of 28 for 6th month
Anuradhapura had a total precipitation of 912 hours with a mean temperature of 28 for 7th month
Anuradhapura had a total precipitation of 1162 hours with a mean temperature of 28 for 8th month
Anuradhapura had a total precipitation of 1615 hours with a mean temperature of 27 for 9th month
Anuradhapura had a total precipitation of 2641 hours with a mean temperature of 26 for 10th month
Anuradhapura had a total precipitation of 3065 hours with a mean temperature of 25 for 11th month
Anuradhapura had a total precipitation of 2449 hours with a mean temperature of 25 for 12th month
Badulla had a total precipitation of 3259 hours with a mean temperature of 21 for 1st month
Badulla had a total precipitation of 2740 hours with a mean temperature of 21 for 2nd month
Badulla had a total precipitation of 1822 hours with a mean temperature of 23 for 3rd month
Badulla had a total precipitation of 1918 hours with a mean temperature of 24 for 4th month
Badulla had a total precipitation of 2063 hours with a mean temperature of 25 for 5th month
Badulla had a total precipitation of 1061 hours with a mean temperature of 25 for 6th month
Badulla had a total precipitation of 1063 hours with a mean temperature of 25 for 7th month
Badulla had a total precipitation of 1247 hours with a mean temperature of 25 for 8th month
Badulla had a total precipitation of 1549 hours with a mean temperature of 24 for 9th month
Badulla had a total precipitation of 2619 hours with a mean temperature of 23 for 10th month
Badulla had a total precipitation of 3335 hours with a mean temperature of 22 for 11th month
Badulla had a total precipitation of 3508 hours with a mean temperature of 22 for 12th month
Batticaloa had a total precipitation of 3301 hours with a mean temperature of 26 for 1st month
Batticaloa had a total precipitation of 2678 hours with a mean temperature of 26 for 2nd month
Batticaloa had a total precipitation of 1880 hours with a mean temperature of 27 for 3rd month
Batticaloa had a total precipitation of 1529 hours with a mean temperature of 28 for 4th month
Batticaloa had a total precipitation of 1529 hours with a mean temperature of 29 for 5th month
Batticaloa had a total precipitation of 738 hours with a mean temperature of 30 for 6th month
Batticaloa had a total precipitation of 728 hours with a mean temperature of 30 for 7th month
Batticaloa had a total precipitation of 806 hours with a mean temperature of 29 for 8th month
```

Task 2 [1] 1 - Implementation:

I have implemented the following Java classes.

MonthlyWeatherDriver.java - Configures and launches the job

```
package com.rgu.bigdata;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

/**
 * Driver for Task 2.1.1: Monthly precipitation and temperature analysis
 */
public class MonthlyWeatherDriver extends Configured implements Tool {

    @Override
    public int run(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: MonthlyWeatherDriver <input path>
<output path>");
            System.err.println(
                    "Example: hadoop jar yourjar.jar
com.rgu.bigdata.MonthlyWeatherDriver /input/weather.csv /output/monthly");
            return -1;
        }

        Configuration conf = getConf();
        Job job = Job.getInstance(conf, "Monthly Weather Analysis - Sri
Lanka");

        job.setJarByClass(MonthlyWeatherDriver.class);

        // Set Mapper and Reducer
        job.setMapperClass(MonthlyWeatherMapper.class);
        job.setReducerClass(MonthlyWeatherReducer.class);

        // Set output types
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // Set input and output paths
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));

        boolean success = job.waitForCompletion(true);
        return success ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
        int exitCode = ToolRunner.run(new Configuration(), new
MonthlyWeatherDriver(), args);
        System.exit(exitCode);
    }
```

MonthlyWeatherMapper.java - Extracts location, month, year, precipitation, temperature

```
package com.rgu.bigdata;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;
import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

/**
 * Task 2.1.1: Calculate total precipitation and mean temperature per district
 * per month over the past decade (2014-2024)
 */
public class MonthlyWeatherMapper extends Mapper<LongWritable, Text, Text, Text> {

    private Text outputKey = new Text();
    private Text outputValue = new Text();
    private Map<String, Integer> columnIndexMap = new HashMap<>();
    private boolean headerProcessed = false;

    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {

        String line = value.toString();
        String[] fields = line.split(",");

        // Process header to build column index map
        if (key.get() == 0) {
            for (int i = 0; i < fields.length; i++) {
                columnIndexMap.put(fields[i].trim(), i);
            }
            headerProcessed = true;
            return;
        }

        if (!headerProcessed) {
            return; // Safety check
        }

        try {
            // Extract fields by column name
            String locationId = fields[columnIndexMap.get("location_id")].trim();
            String dateStr = fields[columnIndexMap.get("date")].trim();
            String temperatureMean = fields[columnIndexMap.get("temperature_2m_mean
(°C)")].trim();
            String precipitationHours = fields[columnIndexMap.get("precipitation_hours
(h)")].trim();

            // Filter out Welimada (location_id=25) and Bandarawela (location_id=26)
            if (locationId.equals("25") || locationId.equals("26")) {
                return; // Skip these cities as they are not districts
            }

            // Parse date to extract month and year
            String[] dateParts = dateStr.split("/");
            if (dateParts.length == 3) {
                String month = dateParts[0]; // Month
                String year = dateParts[2]; // Year
```

```
                // Filter for past decade only (2014-2024)
                int yearInt = Integer.parseInt(year);
                if (yearInt < 2014) {
                    return; // Skip records before 2014
                }

                // Create key: locationId-month (WITHOUT year to aggregate across all
years)
                String keyStr = locationId + "-" + month;
                outputKey.set(keyStr);

                // Create value: precipitation,temperature,count
                String valueStr = precipitationHours + "," + temperatureMean + ",1";
                outputValue.set(valueStr);

                context.write(outputKey, outputValue);
            }

        } catch (Exception e) {
            // Log and skip malformed records
            System.err.println("Error processing line: " + line);
            System.err.println("Error: " + e.getMessage());
        }

    }
}
```

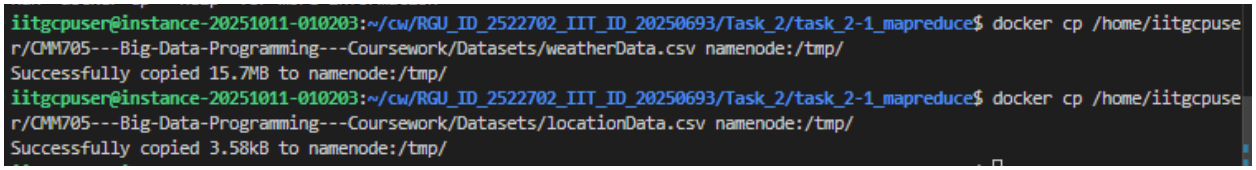
MonthlyWeatherReducer.java - Aggregates data per district, per month

```
package com.rgu.bigdata;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.IOException;
import java.util.*;

/**
 * Task 2.1.1: Reducer for calculating total precipitation and mean temperature
 * per district per month over the past decade (2014-2024)
 *
 */
public class MonthlyWeatherReducer extends Reducer<Text, Text, Text, Text> {

    private Text outputKey = new Text();
    private Text result = new Text();

    // Store all results for sorting
    private List<OutputRecord> outputRecords = new ArrayList<>();

    // Inner class to hold output data
    private static class OutputRecord {
        String locationId;
        int month;
        String cityName;
        double totalPrecipitation;
        double meanTemperature;

        OutputRecord(String locationId, int month, String cityName,
                double totalPrecipitation, double meanTemperature) {
            this.locationId = locationId;
            this.month = month;
            this.cityName = cityName;
            this.totalPrecipitation = totalPrecipitation;
            this.meanTemperature = meanTemperature;
        }
    }

    // Map location_id to city name (loaded dynamically from locationData.csv)
    private Map<String, String> locationMap = new HashMap<>();

    @Override
    protected void setup(Context context) throws IOException, InterruptedException {
        super.setup(context);
        loadLocationData(context);
    }

    /**
     * Load location names from locationData.csv in HDFS
     */
    private void loadLocationData(Context context) throws IOException {
        Configuration conf = context.getConfiguration();
        Path locationPath = new Path("/input/locationData.csv");
        FileSystem fs = FileSystem.get(conf);
```

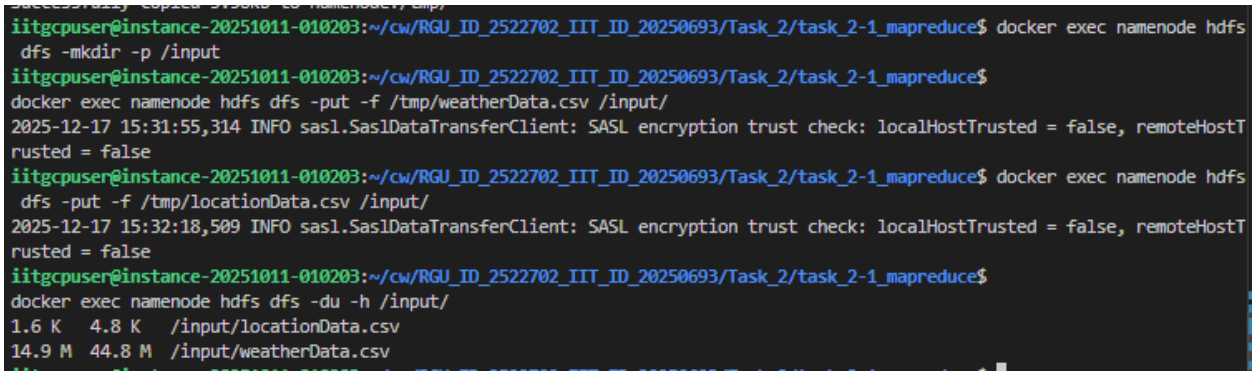
```
        try (BufferedReader br = new BufferedReader(new
InputStreamReader(fs.open(locationPath)))) {
            String line;
            boolean isHeader = true;

            while ((line = br.readLine()) != null) {
                if (isHeader) {
                    isHeader = false;
                    continue; // Skip header line
                }

                String[] fields = line.split(",");
                if (fields.length >= 8) {
                    String locationId = fields[0].trim();
                    String cityName = fields[7].trim();
                    locationMap.put(locationId, cityName);
                }
            }
            System.out.println("Loaded " + locationMap.size() + " locations from
locationData.csv");
        } catch (IOException e) {
            System.err.println("Error reading locationData.csv: " + e.getMessage());
            throw e;
        }
    }

    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

        double totalPrecipitation = 0.0;
        double totalTemperature = 0.0;
        int count = 0;

        // Aggregate values for this location-month-year
        for (Text value : values) {
            String[] parts = value.toString().split(",");

            if (parts.length == 3) {
                try {
                    double precip = Double.parseDouble(parts[0]);
                    double temp = Double.parseDouble(parts[1]);
                    int recordCount = Integer.parseInt(parts[2]);

                    totalPrecipitation += precip;
                    totalTemperature += temp;
                    count += recordCount;
                } catch (NumberFormatException e) {
                    System.err.println("Error parsing values: " + value.toString());
                }
            }
        }

        // Calculate mean temperature
        double meanTemperature = count > 0 ? totalTemperature / count : 0.0;

        // Parse key to get location_id and month (NO year - aggregated across all
        // years)
        String[] keyParts = key.toString().split("-");
```

Task 2 [1] 1 – Execution Steps:

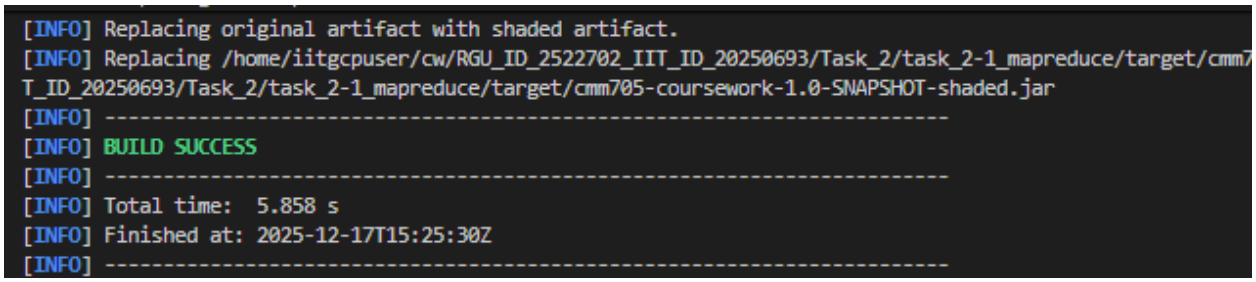
1. Copied datasets to namenode container.



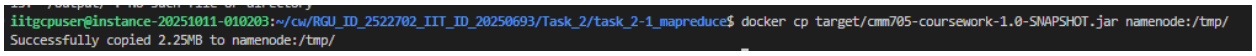
2. Uploaded copied data into HDFS



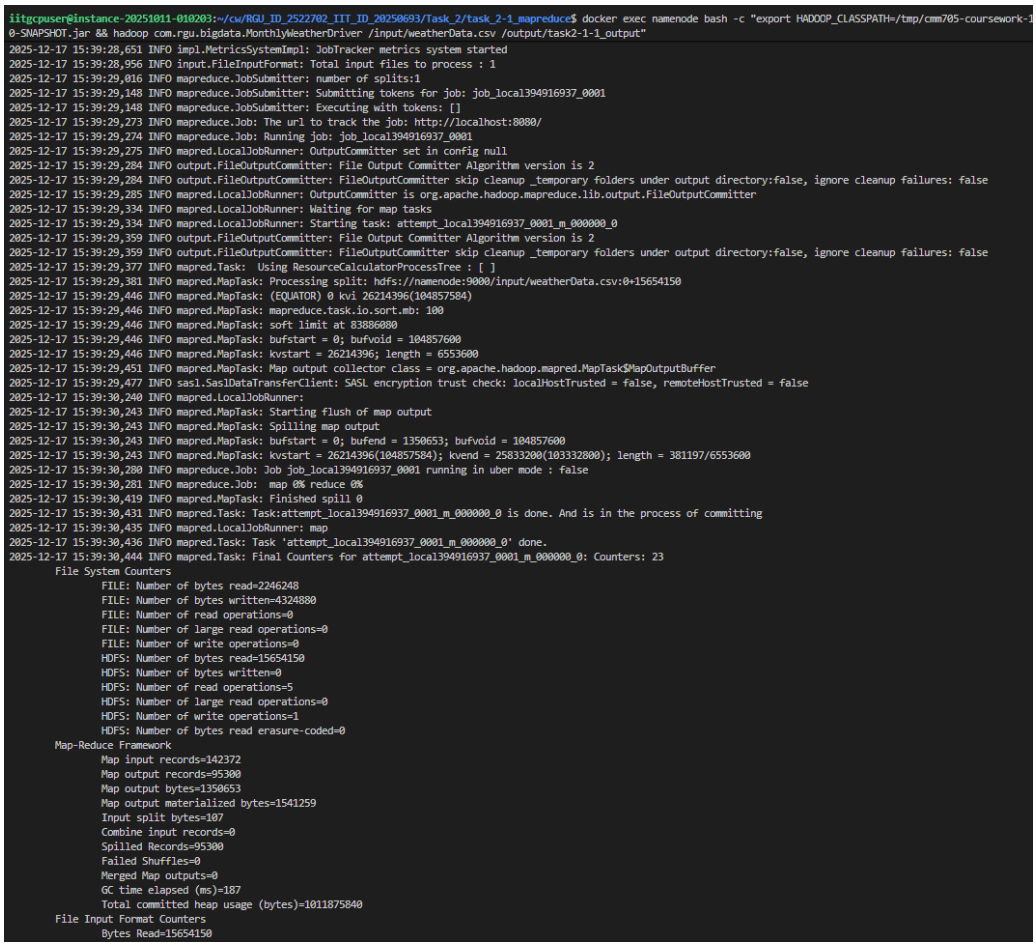
3. Built the MapReduce JAR

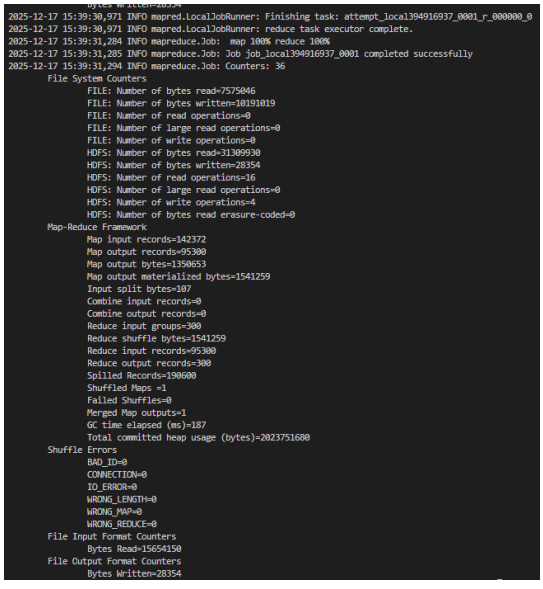


4. Copied the generated JAR into namenode.



5. Ran MapReduce Job MonthlyWeatherDriver.

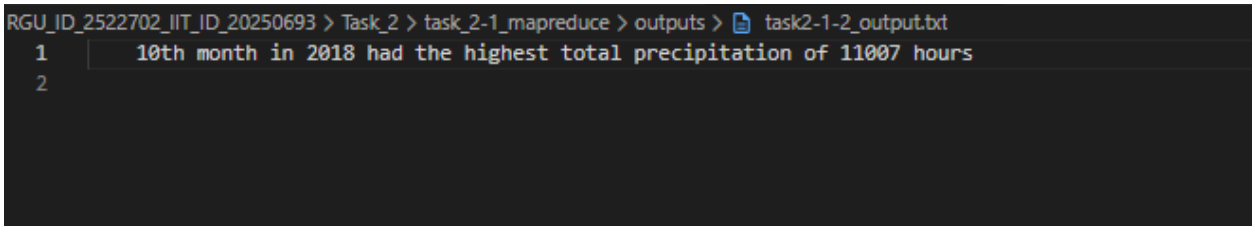




- MonthlyWeatherDriver job has executed successfully, and results had saved into a txt file in HDFS.

[Task 2] 1. 2 Hadoop MapReduce Analysis

Task 2 [1] 2 – Hadoop MapReduce Analysis - Final Output:



Task 2 [1] 2 – Implementation:

- MaxPrecipitationDriver.java - Configures and launches the job

```
package com.rgu.bigdata;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

/**
 * Driver for Task 2.1.2: Find the month and year with the highest total
 * precipitation
 * This job aggregates precipitation data by month and identifies the month
 * with the maximum total precipitation across all districts in Sri Lanka.
 */
public class MaxPrecipitationDriver extends Configured implements Tool {

    @Override
    public int run(String[] args) throws Exception {
        if (args.length != 2) {
            System.err.println("Usage: MaxPrecipitationDriver <input_path>
<output_path>");
            System.err.println("Example: hadoop jar bigdata-weather-1.0.jar
" +
                    "com.rgu.bigdata.MaxPrecipitationDriver " +
                    "/data/weatherData.csv /output/task2_1_2");
            return -1;
        }

        Configuration conf = getConf();
        Job job = Job.getInstance(conf, "Task 2.1.2 - Maximum Precipitation
Month");

        job.setJarByClass(MaxPrecipitationDriver.class);
        job.setMapperClass(MaxPrecipitationMapper.class);
        job.setReducerClass(MaxPrecipitationReducer.class);

        // Output key/value types
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(Text.class);

        // Input/output paths
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
```

```
        // Set number of reducers to 1 to ensure global maximum is found
        job.setNumReduceTasks(1);

        System.out.println("============================================");
        System.out.println("Task 2.1.2: Maximum Precipitation Analysis");
        System.out.println("============================================");
        System.out.println("Input Path: " + args[0]);
        System.out.println("Output Path: " + args[1]);
        System.out.println("Mapper: MaxPrecipitationMapper");
        System.out.println("Reducer: MaxPrecipitationReducer");
        System.out.println("============================================");

        boolean success = job.waitForCompletion(true);

        if (success) {
            System.out.println("\n✓ Job completed successfully!");
            System.out.println("Check output at: " + args[1]);
            System.out.println("Look for '==== MAXIMUM PRECIPITATION MONTH
====' in the output file");
        } else {
            System.err.println("\n✗ Job failed!");
        }

        return success ? 0 : 1;
    }

    public static void main(String[] args) throws Exception {
```

- MaxPrecipitationMapper.java - Extracts month, year, precipitation

```
package com.rgu.bigdata;

import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

/**
 * Mapper for Task 2.1.2: Find the month and year with the highest total
 * precipitation
 * This mapper extracts the month-year and precipitation from each record,
 * allowing the reducer to find the month with maximum total precipitation.
 */
public class MaxPrecipitationMapper extends Mapper<LongWritable, Text, Text,
Text> {

    private Text outputKey = new Text();
    private Text outputValue = new Text();
    private Map<String, Integer> columnIndexMap = new HashMap<>();
    private boolean headerProcessed = false;

    @Override
    protected void map(LongWritable key, Text value, Context context)
            throws IOException, InterruptedException {

        String line = value.toString();
        String[] fields = line.split(",");

        // Process header to build column index map
        if (key.get() == 0) {
            for (int i = 0; i < fields.length; i++) {
                columnIndexMap.put(fields[i].trim(), i);
            }
            headerProcessed = true;
            return;
        }

        if (!headerProcessed) {
            return; // Safety check
        }

        try {
            // Extract fields by column name
            String locationId =
fields[columnIndexMap.get("location_id")].trim();
            String dateStr = fields[columnIndexMap.get("date")].trim();
            String precipitationHours =
fields[columnIndexMap.get("precipitation_hours (h)")].trim();

            // Parse date to extract month and year
            String[] dateParts = dateStr.split("/");
            if (dateParts.length == 3) {
```

```
                String month = dateParts[0]; // Month
                String year = dateParts[2]; // Year

        // Create key: month-year
                String yearMonth = month + "-" + year;
                outputKey.set(yearMonth);

                // Emit precipitation value
                outputValue.set(precipitationHours);
                context.write(outputKey, outputValue);
            }
        } catch (Exception e) {
            // Skip records with parsing errors
            System.err.println("Error parsing line: " + line);
        }
    }
}
```

- MaxPrecipitationReducer.java - Finds month with highest precipitation

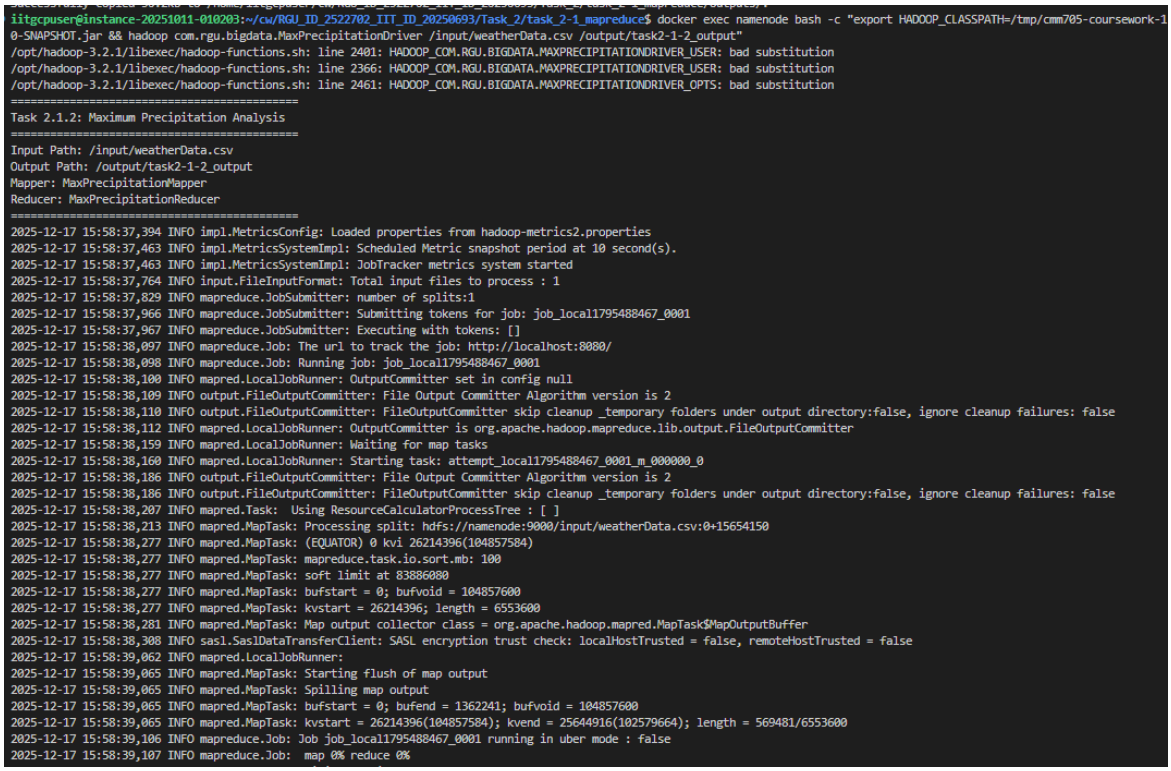
```
package com.rgu.bigdata;

import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

import java.io.IOException;

/**
 * Reducer for Task 2.1.2: Find the month and year with the highest total
 * precipitation
 * This reducer aggregates precipitation for each month and tracks the
maximum.
 */
public class MaxPrecipitationReducer extends Reducer<Text, Text, Text, Text>
{

    // Global variable to track maximum precipitation month
    private double globalMaxPrecipitation = Double.MIN_VALUE;
    private String maxMonth = "";

    @Override
    protected void reduce(Text key, Iterable<Text> values, Context context)
            throws IOException, InterruptedException {

        double totalPrecipitation = 0.0;
        int count = 0;
```

```
        // Aggregate precipitation for this month-year
        for (Text value : values) {
            try {
                double precipitation = Double.parseDouble(value.toString());
                totalPrecipitation += precipitation;
                count++;
            } catch (NumberFormatException e) {
                System.err.println("Error parsing precipitation value: " +
value.toString());
            }
        }

        // Check if this is the global maximum
        if (totalPrecipitation > globalMaxPrecipitation) {
            globalMaxPrecipitation = totalPrecipitation;
            maxMonth = key.toString();
        }

        // Emit all months with their totals
        String output = String.format("Total Precipitation: %.2f hours (from
%d records)",
                totalPrecipitation, count);
        // context.write(key, new Text(output));
    }

    @Override
    protected void cleanup(Context context) throws IOException,
InterruptedException {
        // Emit the maximum month summary at the end
        if (!maxMonth.isEmpty()) {
            String[] parts = maxMonth.split("-");
            String month = parts[0];
            String year = parts[1];

            // Format: "2nd month in 2019 had the highest total precipitation
of 300 hr"
            String result = String.format("%s month in %s had the highest
total precipitation of %.0f hours",
                    getOrdinal(Integer.parseInt(month)), year,
globalMaxPrecipitation);

            context.write(new Text(""), new Text(result));
        }
    }

    /**
     * Helper method to convert month number to ordinal (1st, 2nd, 3rd, etc.)
     */
    private String getOrdinal(int month) {
        if (month >= 11 && month <= 13) {
            return month + "th";
        }
```

```
        switch (month % 10) {
            case 1:
                return month + "st";
            case 2:
                return month + "nd";
            case 3:
                return month + "rd";
            default:
                return month + "th";
        }
    }
}
```

Task 2 [1] 2 – Execution Steps:

1. After copying the input datasets into HDFS as in previous step, executed the MapReduce MaxPrecipitationDriver job.



```
        File System Counters
                FILE: Number of bytes read=7786506
                FILE: Number of bytes written=10513325
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=31308300
                HDFS: Number of bytes written=71
                HDFS: Number of read operations=15
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=4
                HDFS: Number of bytes read erasure-coded=0
        Map-Reduce Framework
                Map input records=142372
                Map output records=142371
                Map output bytes=1362241
                Map output materialized bytes=1646989
                Input split bytes=107
                Combine input records=0
                Combine output records=0
                Reduce input groups=174
                Reduce shuffle bytes=1646989
                Reduce input records=142371
                Reduce output records=1
                Spilled Records=284742
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=166
                Total committed heap usage (bytes)=2048917504
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=15654150
        File Output Format Counters
                Bytes Written=71

? Job completed successfully!
Check output at: /output/task2-1-2_output
```

[Task 2] 2. Hive Analysis

Task 2 [2] 1 – Hive Analysis - Final Output:

```
city_name       avg_max_temp
Nuwara Eliya    20.26
Bandarawela     24.58
Welimada        25.28
Badulla 27.64
Kandy   27.73
Matara  28.5
Galle   28.76
Ratnapura       28.85
Matale  28.88
Kalutara        29.18
```

Task 2 [2] 1 - Implementation steps:

1. Copied Datasets to namenode.

```
iitgcpuser@instance-20251011-010203:~/CMM705---Big-Data-Programming---Coursework/RGU_ID_2522702_IIT_ID_20250693/Task_2/task_2-1_mapreduce$ sudo docker cp /home/iitgcpuser/CMM705-
-Big-Data-Programming---Coursework/Datasets/locationData.csv namenode:/tmp/
Successfully copied 3.58kB to namenode:/tmp/
iitgcpuser@instance-20251011-010203:~/CMM705---Big-Data-Programming---Coursework/RGU_ID_2522702_IIT_ID_20250693/Task_2/task_2-1_mapreduce$ sudo docker cp /home/iitgcpuser/CMM705-
-Big-Data-Programming---Coursework/Datasets/weatherData.csv namenode:/tmp/
Successfully copied 15.7MB to namenode:/tmp/
```

2. Entered the Hadoop namenode container, created directories and uploaded the copied datasets to HDFS.

```
iitgcpuser@instance-20251011-010203:~/CMM705---Big-Data-Programming---Coursework/RGU_ID_2522702_IIT_ID_20250693/Task_2/task_2-1_mapreduce$ sudo docker exec -it namenode bash
root@dd77cd0c8e2c:/# hdfs dfs -mkdir -p /user/hive/warehouse/weather_analytics.db/location
root@dd77cd0c8e2c:/# hdfs dfs -mkdir -p /user/hive/warehouse/weather_analytics.db/weather
root@dd77cd0c8e2c:/# hdfs dfs -put /tmp/locationData.csv /user/hive/warehouse/weather_analytics.db/location/
2025-12-17 11:54:29,997 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
root@dd77cd0c8e2c:/# hdfs dfs -put /tmp/weatherData.csv /user/hive/warehouse/weather_analytics.db/weather/
2025-12-17 11:54:49,890 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
```

Data had uploaded correctly to HDFS.

```
root@dd77cd0c8e2c:/# hdfs dfs -ls /user/hive/warehouse/weather_analytics.db/location
Found 1 items
-rw-r--r--   3 root supergroup       1630 2025-12-17 11:54 /user/hive/warehouse/weather_analytics.db/location/locationData.csv
root@dd77cd0c8e2c:/# hdfs dfs -ls /user/hive/warehouse/weather_analytics.db/weather
Found 1 items
-rw-r--r--   3 root supergroup   15654150 2025-12-17 11:54 /user/hive/warehouse/weather_analytics.db/weather/weatherData.csv
root@dd77cd0c8e2c:/#
```

```
root@dd77cd0c8e2c:/# hdfs dfs -cat /user/hive/warehouse/weather_analytics.db/location/locationData.csv | head -5
2025-12-17 11:57:26,347 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
location_id,latitude,longitude,elevation,utc_offset_seconds,timezone,timezone_abbreviation,city_name
0,6.924429,79.90725,4,19800,Asia/Colombo,530,Colombo
1,7.0650263,79.96622,19,19800,Asia/Colombo,530,Gampaha
2,6.572935,80.02519,5,19800,Asia/Colombo,530,Kalutara
3,7.275923,80.62659,499,19800,Asia/Colombo,530,Kandy
root@dd77cd0c8e2c:/# hdfs dfs -cat /user/hive/warehouse/weather_analytics.db/weather/weatherData.csv | head -5
```

```
root@dd77cd0c8e2c:/# hdfs dfs -cat /user/hive/warehouse/weather_analytics.db/weather/weatherData.csv | head -5

2025-12-17 11:57:35,227 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
location_id,date,weather_code (wmo code),temperature_2m_max (°C),temperature_2m_min (°C),temperature_2m_mean (°C),apparent_temperature_max (°C),apparent_temperature_min (°C),appar
ent_temperature_mean (°C),daylight_duration (s),sunshine_duration (s),precipitation_sum (mm),rain_sum (mm),precipitation_hours (h),wind_speed_10m_max (km/h),wind_gusts_10m_max (km
/h),wind_direction_10m_dominant (°),shortwave_radiation_sum (MJ/m²),et0_fao_evapotranspiration (mm),sunrise,sunset
0,1/1/2010,1,30.1,22.6,26.0,34.5,25.0,29.0,42220.2,38905.73,0.0,0.0,0.0,12.2,27.4,19,20.92,4.61,06:22,18:05
0,1/2/2010,51,30.1,23.7,26.3,33.9,26.1,29.7,42225.71,37451.01,0.1,0.1,1,13.0,27.0,24,17.71,3.91,06:22,18:06
0,1/3/2010,51,29.6,23.1,26.0,34.5,26.2,29.9,42231.68,33176.43,0.6,0.6,3,12.3,27.4,17,17.76,3.66,06:22,18:06
0,1/4/2010,2,28.9,23.1,25.7,31.7,26.1,28.4,42238.11,38289.2,0.0,0.0,0.0,17.0,34.6,357,16.5,3.75,06:23,18:07
```

3. Connected to Hive server and created a Database and Tables in Hive.

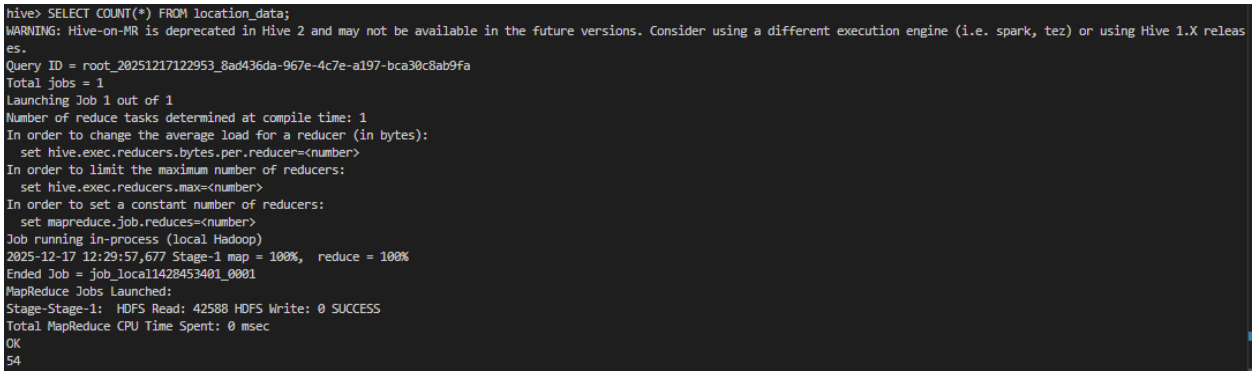
```
hive> SHOW DATABASES;
OK
default
weather_analytics
Time taken: 0.011 seconds, Fetched: 2 row(s)
hive> USE weather_analytics;
OK
Time taken: 0.016 seconds
hive> SHOW TABLES;
OK
location_data
weather_data
Time taken: 0.025 seconds, Fetched: 2 row(s)
```

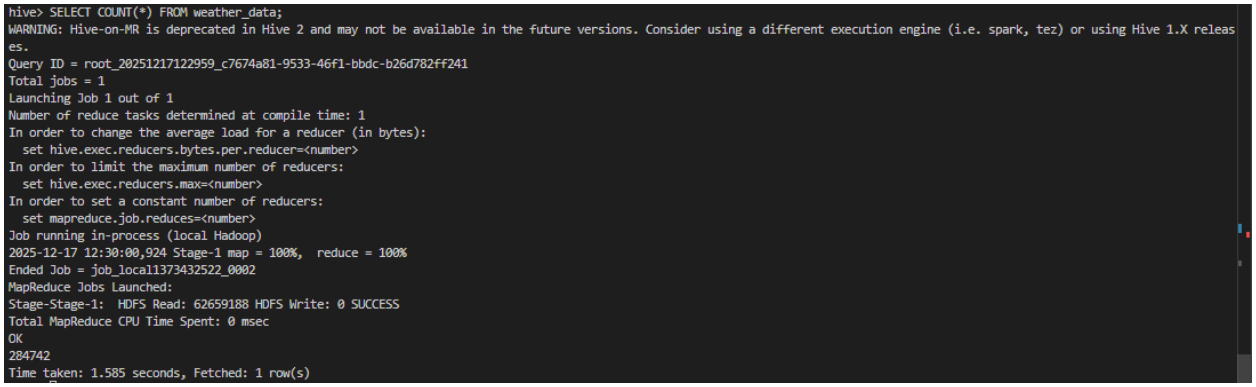
4. Loaded data in HDFS into the tables created.

```
hive> LOAD DATA INPATH '/user/hive/warehouse/weather_analytics.db/location/locationData.csv' INTO TABLE location_data;
Loading data to table default.location_data
OK
Time taken: 1.017 seconds
hive> LOAD DATA INPATH '/user/hive/warehouse/weather_analytics.db/weather/weatherData.csv' INTO TABLE weather_data;
Loading data to table default.weather_data
OK
Time taken: 0.423 seconds
```

```
hive> DESCRIBE location_data;
OK
location_id             int                     Unique location identifier
latitude                double                  Latitude coordinate
longitude               double                  Longitude coordinate
elevation               double                  Elevation above sea level in meters
utc_offset_seconds      int                     UTC offset in seconds
timezone                string                  Timezone name
timezone_abbreviation   string                  Timezone abbreviation
city_name               string                  Name of the city
Time taken: 0.046 seconds, Fetched: 8 row(s)
hive> DESCRIBE weather_data;
OK
location_id             int                     Location identifier matching locationData
date                    string                  Date in format M/D/YYYY
weather_code            int                     WMO weather code
temperature_2m_max      double                  Maximum temperature in ℃
temperature_2m_min      double                  Minimum temperature in ℃
temperature_2m_mean     double                  Mean temperature in ℃
```

Successfully loaded data could be seen in both tables.





5. Executed the following Hive query to Rank the top 10 most temperate cities across the dataset.

```
SELECT
    l.city_name AS city_name,
    ROUND(AVG(w.temperature_2m_max), 2) AS avg_max_temp
FROM
    weather_data w
JOIN
    location_data l
    ON w.location_id = l.location_id
WHERE
    w.temperature_2m_max IS NOT NULL
GROUP BY
    l.city_name
ORDER BY
    avg_max_temp ASC
LIMIT 10;
```

The following configuration was set in order to display column names in the output.

set hive.cli.print.header=true;

```
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2025-12-17 12:37:31,680 Stage-2 map = 100%,  reduce = 100%
Ended Job = job_local2049546362_0005
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2025-12-17 12:37:32,953 Stage-3 map = 100%,  reduce = 100%
Ended Job = job_local1680950613_0006
MapReduce Jobs Launched:
Stage-Stage-2:  HDFS Read: 187892388 HDFS Write: 0 SUCCESS
Stage-Stage-3:  HDFS Read: 187892388 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
city_name       avg_max_temp
Nuwara Eliya    20.26
Bandarawela     24.58
Welimada        25.28
Badulla 27.64
Kandy   27.73
Matara  28.5
Galle   28.76
Ratnapura       28.85
Matale  28.88
Kalutara        29.18
Time taken: 11.714 seconds, Fetched: 10 row(s)
```

Task 2 [2] 1 – Hive Analysis - Final Output:

```
city_name       season  avg_evapotranspiration
Ampara  Maha (Sep-Mar)  3.8
Ampara  Yala (Apr-Aug)  4.87
Anuradhapura    Maha (Sep-Mar)  3.89
Anuradhapura    Yala (Apr-Aug)  4.76
Badulla Maha (Sep-Mar)  3.55
Badulla Yala (Apr-Aug)  4.41
Bandarawela     Maha (Sep-Mar)  3.43
Bandarawela     Yala (Apr-Aug)  3.92
Batticaloa      Maha (Sep-Mar)  3.92
Batticaloa      Yala (Apr-Aug)  5.03
Colombo Maha (Sep-Mar)  3.81
Colombo Yala (Apr-Aug)  3.64
Galle   Maha (Sep-Mar)  3.87
Galle   Yala (Apr-Aug)  3.92
Gampaha Maha (Sep-Mar)  3.88
Gampaha Yala (Apr-Aug)  3.59
Hambantota      Maha (Sep-Mar)  4.37
Hambantota      Yala (Apr-Aug)  4.95
Jaffna  Maha (Sep-Mar)  4.31
Jaffna  Yala (Apr-Aug)  5.2
Kalutara        Maha (Sep-Mar)  3.76
Kalutara        Yala (Apr-Aug)  3.65
```

Task 2 [2] 1 - Implementation steps:

Followed the same steps from Step 1 to 4 as in above task.

1. Since the Date format in the provided dataset is M/D/YYYY or MM/DD/YYYY, first I created a temporary table with parsed dates. We can use this to extract month from the date strig.

```
hive> SELECT * FROM weather_with_month LIMIT 10;
OK
0       1/1/2010        4.61    1
0       1/2/2010        3.91    1
0       1/3/2010        3.66    1
0       1/4/2010        3.75    1
0       1/5/2010        5.0     1
0       1/6/2010        4.95    1
0       1/7/2010        4.38    1
0       1/8/2010        2.98    1
0       1/9/2010        3.34    1
0       1/10/2010       4.15    1
Time taken: 0.189 seconds, Fetched: 10 row(s)
hive> SELECT COUNT(*) FROM weather_with_month;
OK
284742
Time taken: 0.257 seconds, Fetched: 1 row(s)
```

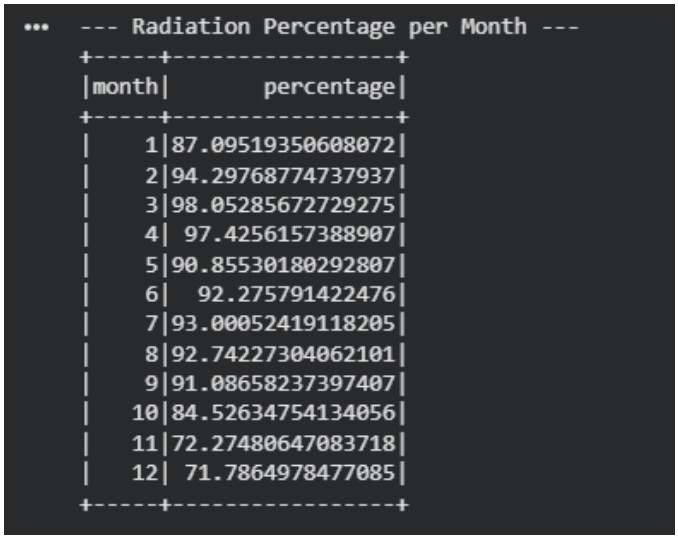
2. Executed the following Hive query to calculate the average evapotranspiration for each major agricultural season in each district over the years.

```
SELECT
    l.city_name AS city_name,
    CASE
        WHEN w.month_num IN (9, 10, 11, 12, 1, 2, 3) THEN 'Maha (Sep-Mar)'
        WHEN w.month_num IN (4, 5, 6, 7, 8) THEN 'Yala (Apr-Aug)'
    END AS season,
    ROUND(AVG(w.et0_fao_evapotranspiration), 2) AS avg_evapotranspiration
FROM
    weather_with_month w
JOIN
    location_data l
    ON w.location_id = l.location_id
WHERE
    w.month_num IN (1,2,3,4,5,6,7,8,9,10,11,12)
GROUP BY
    l.city_name,
    CASE
        WHEN w.month_num IN (9, 10, 11, 12, 1, 2, 3) THEN 'Maha (Sep-Mar)'
        WHEN w.month_num IN (4, 5, 6, 7, 8) THEN 'Yala (Apr-Aug)'
    END
ORDER BY
    city_name,
    season;
```

```
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2025-12-17 13:20:31,569 Stage-2 map = 0%,  reduce = 0%
2025-12-17 13:20:32,580 Stage-2 map = 100%,  reduce = 100%
Ended Job = job_local696758915_0002
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2025-12-17 13:20:33,920 Stage-3 map = 100%,  reduce = 100%
Ended Job = job_local137366302_0003
MapReduce Jobs Launched:
Stage-Stage-2:  HDFS Read: 73855690 HDFS Write: 11230898 SUCCESS
Stage-Stage-3:  HDFS Read: 73855690 HDFS Write: 11230898 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
Ampara  Maha (Sep-Mar)  3.8
Ampara  Yala (Apr-Aug)  4.87
Anuradhapura    Maha (Sep-Mar)  3.89
Anuradhapura    Yala (Apr-Aug)  4.76
Badulla Maha (Sep-Mar)  3.55
Badulla Yala (Apr-Aug)  4.41
Bandarawela     Maha (Sep-Mar)  3.43
Bandarawela     Yala (Apr-Aug)  3.92
Batticaloa      Maha (Sep-Mar)  3.92
Batticaloa      Yala (Apr-Aug)  5.03
Colombo Maha (Sep-Mar)  3.81
Colombo Yala (Apr-Aug)  3.64
Galle   Maha (Sep-Mar)  3.87
Galle   Yala (Apr-Aug)  3.92
Gampaha Maha (Sep-Mar)  3.88
Gampaha Yala (Apr-Aug)  3.59
Hambantota      Maha (Sep-Mar)  4.37
Hambantota      Yala (Apr-Aug)  4.95
Jaffna  Maha (Sep-Mar)  4.31
```

[Task 2] 3. Spark Analysis

Task 2 [3] 1 – Spark Analysis - Final Output:



```
--- Radiation Percentage per Month ---
+-----+-----------------+
|month|       percentage|
+-----+-----------------+
|    1|87.09519350608072|
|    2|94.29768774737937|
|    3|98.05285672729275|
|    4| 97.4256157388907|
|    5|90.85530180292807|
|    6|  92.275791422476|
|    7|93.00052419118205|
|    8|92.74227304062101|
|    9|91.08658237397407|
|   10|84.52634754134056|
|   11|72.27480647083718|
|   12| 71.7864978477085|
+-----+-----------------+
```

Task 2 [3] 1 - Implementation:

Pre-processed data to be used in spark analysis.

```
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.window import Window

# Initialize Spark Session
spark = SparkSession.builder.appName("SL_Weather_Analysis").getOrCreate()

# Load Datasets
loc_df =
spark.read.csv("/content/drive/MyDrive/CW_DATASETS/locationData.csv",
header=True, inferSchema=True)
weather_df =
spark.read.csv("/content/drive/MyDrive/CW_DATASETS/weatherData.csv",
header=True, inferSchema=True)

# Preprocessing
# Convert date string to DateType (Format in CSV is M/d/yyyy based on
inspection)
weather_df = weather_df.withColumn("date", F.to_date(F.col("date"),
"M/d/yyyy"))
weather_df = weather_df.withColumn("year", F.year("date")) \
                       .withColumn("month", F.month("date"))

# Join with Location Data to get City Names
df = weather_df.join(loc_df, "location_id")

# Cache df for better performance on multiple queries
df.cache()
```

I mainly focused on quantifying how much of the monthly shortwave radiation exceeded a threshold of **15 MJ/m²** across all districts.

- Sum of shortwave radiation values was calculated for each month.
- Radiation values greater than 15 MJ/m² were isolated and summed separately.
- The ratio of high radiation to total radiation was computed for each month.
- This percentage represents the share of monthly radiation that surpassed the 15 MJ/m² threshold.

```
# Calculate Total Radiation and High Radiation (>15) per month
q1_result = df.groupBy("month").agg(
    F.sum("shortwave_radiation_sum (MJ/m²)").alias("total_radiation"),
    F.sum(F.when(F.col("shortwave_radiation_sum (MJ/m²)") > 15,
                 F.col("shortwave_radiation_sum
(MJ/m²)")).otherwise(0)).alias("high_radiation")
).withColumn("percentage", (F.col("high_radiation") /
F.col("total_radiation")) * 100) \
 .orderBy("month","percentage")

print("--- Radiation Percentage per Month ---")
q1_result.select("month", "percentage").show()
```

Task 2 [3] 2 – Spark Analysis - Final Output:

```
--- Weekly Max Temps for Hottest Months ---
+----+-----+----+---------------+
|year|month|week|weekly_max_temp|
+----+-----+----+---------------+
|2010|    3|   9|           36.1|
|2010|    3|  10|           36.7|
|2010|    3|  11|           37.2|
|2010|    3|  12|           36.6|
|2010|    3|  13|           33.9|
|2011|    6|  22|           35.1|
|2011|    6|  23|           35.2|
|2011|    6|  24|           35.5|
|2011|    6|  25|           36.4|
|2011|    6|  26|           37.0|
|2012|    5|  18|           35.4|
|2012|    5|  19|           37.1|
|2012|    5|  20|           37.4|
|2012|    5|  21|           36.2|
|2012|    5|  22|           36.4|
|2013|    4|  14|           36.5|
|2013|    4|  15|           36.0|
|2013|    4|  16|           36.2|
|2013|    4|  17|           34.8|
|2013|    4|  18|           34.8|
+----+-----+----+---------------+
only showing top 20 rows
```

Task 2 [3] 2 - Implementation:

- For each year, the average maximum temperature was calculated across all months.

- The month with the highest mean temperature was selected as the “hottest month.”
- The original dataset was joined with the hottest-month results to isolate records belonging only to those months.
- Within each hottest month, weekly groupings were created using the calendar week number.
- For each week, the maximum daily temperature was computed.

```
# Identify hottest month for each year (based on Mean Temp)
monthly_stats = df.groupBy("year", "month").agg(F.mean("temperature_2m_max
(°C)").alias("monthly_avg_temp"))

window_year =
Window.partitionBy("year").orderBy(F.col("monthly_avg_temp").desc())

hottest_months = monthly_stats.withColumn("rank",
F.row_number().over(window_year)) \
    .filter(F.col("rank") == 1) \
    .select(F.col("year").alias("hot_year"),
F.col("month").alias("hot_month"))

# Join back to original data to filter for these months
df_hottest = df.join(hottest_months,
                     (df.year == hottest_months.hot_year) & (df.month ==
hottest_months.hot_month))

# Calculate Weekly Max Temperature (using weekofyear)
q2_result = df_hottest.withColumn("week", F.weekofyear("date")) \
    .groupBy("year", "month", "week") \
    .agg(F.max("temperature_2m_max (°C)").alias("weekly_max_temp")) \
    .orderBy("year", "month", "week")

print("--- Weekly Max Temps for Hottest Months ---")
q2_result.show()
```

[Task 3] Performing Machine Learning model using Spark MLlib

Step 1: Environment Setup

```
from pyspark.sql import SparkSession

spark = SparkSession.builder \
    .appName("Weather Analysis") \
    .config("spark.driver.memory", "4g") \
    .getOrCreate()

print("Spark session created successfully!")
print(f"Spark version: {spark.version}")
```

```
Spark session created successfully!
Spark version: 4.0.1
```

Step 2: Data Loading

```
# Load both datasets
# Location dataset contains district information
location_df = spark.read.csv("/content/drive/MyDrive/CW_DATASETS/locationData.csv",
header=True, inferSchema=True)

# Weather dataset contains meteorological data
weather_df = spark.read.csv("/content/drive/MyDrive/CW_DATASETS/weatherData.csv",
header=True, inferSchema=True)

print(f"Location dataset loaded! Total records: {location_df.count():,}")
print(f"Number of columns: {len(location_df.columns)}")
print(f"\nWeather dataset loaded! Total records: {weather_df.count():,}")
print(f"Number of columns: {len(weather_df.columns)}")
```

```
Location dataset loaded! Total records: 27
Number of columns: 8

Weather dataset loaded! Total records: 142,371
Number of columns: 21
```

Step 3: Initial Data Exploration

```
# Check the location dataset structure
print("Location Dataset Schema:")
location_df.printSchema()
print("\nSample location data:")
location df.show(5, truncate=False)
```

```
Location Dataset Schema:
root
 |-- location_id: integer (nullable = true)
 |-- latitude: double (nullable = true)
 |-- longitude: double (nullable = true)
 |-- elevation: integer (nullable = true)
 |-- utc_offset_seconds: integer (nullable = true)
 |-- timezone: string (nullable = true)
 |-- timezone_abbreviation: integer (nullable = true)
 |-- city_name: string (nullable = true)


Sample location data:
+-----------+---------+---------+---------+------------------+------------+---------------------+---------+
|location_id|latitude |longitude|elevation|utc_offset_seconds|timezone    |timezone_abbreviation|city_name|
+-----------+---------+---------+---------+------------------+------------+---------------------+---------+
|0          |6.924429 |79.90725 |4        |19800             |Asia/Colombo|530                  |Colombo  |
|1          |7.0650263|79.96622 |19       |19800             |Asia/Colombo|530                  |Gampaha  |
|2          |6.572935 |80.02519 |5        |19800             |Asia/Colombo|530                  |Kalutara |
|3          |7.275923 |80.62659 |499      |19800             |Asia/Colombo|530                  |Kandy    |
|4          |7.4868193|80.52632 |362      |19800             |Asia/Colombo|530                  |Matale   |
+-----------+---------+---------+---------+------------------+------------+---------------------+---------+
only showing top 5 rows
```

```
# Check the weather dataset schema
print("Weather Dataset Schema:")
weather_df.printSchema()
```

```
Weather Dataset Schema:
root
 |-- location_id: integer (nullable = true)
 |-- date: string (nullable = true)
 |-- weather_code (wmo code): integer (nullable = true)
 |-- temperature_2m_max (°C): double (nullable = true)
 |-- temperature_2m_min (°C): double (nullable = true)
 |-- temperature_2m_mean (°C): double (nullable = true)
 |-- apparent_temperature_max (°C): double (nullable = true)
 |-- apparent_temperature_min (°C): double (nullable = true)
 |-- apparent_temperature_mean (°C): double (nullable = true)
 |-- daylight_duration (s): double (nullable = true)
 |-- sunshine_duration (s): double (nullable = true)
 |-- precipitation_sum (mm): double (nullable = true)
 |-- rain_sum (mm): double (nullable = true)
 |-- precipitation_hours (h): integer (nullable = true)
 |-- wind_speed_10m_max (km/h): double (nullable = true)
 |-- wind_gusts_10m_max (km/h): double (nullable = true)
 |-- wind_direction_10m_dominant (°): integer (nullable = true)
 |-- shortwave_radiation_sum (MJ/m²): double (nullable = true)
 |-- et0_fao_evapotranspiration (mm): double (nullable = true)
 |-- sunrise: timestamp (nullable = true)
 |-- sunset: timestamp (nullable = true)
```

```
print("Sample weather data:")
weather df.show(5, truncate=False)
```

```
Sample weather data:
+-----------+-------+-------------------------+-----------------------+-----------------------+------------------------+-------------------------------+-------------------------------+--------------------------------+-----------------------+
|location_id|date   |weather_code (wmo code)  |temperature_2m_max (°C)|temperature_2m_min (°C)|temperature_2m_mean (°C)|apparent_temperature_max (°C)  |apparent_temperature_min (°C)  |apparent_temperature_mean (°C)  |daylight_durat
+-----------+-------+-------------------------+-----------------------+-----------------------+------------------------+-------------------------------+-------------------------------+--------------------------------+-----------------------+
|0          |1/1/2010|1                       |30.1                   |22.6                   |26.0                    |34.5                           |25.0                           |29.0                            |42220.2
|0          |1/2/2010|51                      |30.1                   |23.7                   |26.3                    |33.9                           |26.1                           |29.7                            |42225.71
|0          |1/3/2010|51                      |29.6                   |23.1                   |26.0                    |34.5                           |26.2                           |29.9                            |42231.68
|0          |1/4/2010|2                       |28.9                   |23.1                   |25.7                    |31.7                           |26.1                           |28.4                            |42238.11
|0          |1/5/2010|1                       |28.1                   |21.3                   |24.6                    |30.0                           |22.9                           |26.2                            |42244.99
+-----------+-------+-------------------------+-----------------------+-----------------------+------------------------+-------------------------------+-------------------------------+--------------------------------+-----------------------+
only showing top 5 rows
```

```
# Get basic statistics for numerical columns
print("Summary statistics:")
weather_df.describe().show()
```

```
Summary statistics:
+-------+-----------------+-------+-------------------------+-----------------------+-----------------------+------------------------+-------------------------------+-------------------------------+--------------------------------+
|summary|      location_id|   date|weather_code (wmo code)  |temperature_2m_max (°C)|temperature_2m_min (°C)|temperature_2m_mean (°C)|apparent_temperature_max (°C)  |apparent_temperature_min (°C)  |apparent_temperature_mean (°C)  |
+-------+-----------------+-------+-------------------------+-----------------------+-----------------------+------------------------+-------------------------------+-------------------------------+--------------------------------+
|  count|           142371| 142371|                   142371|                 142371|                 142371|                  142371|                         142371|                         142371|                          142371|
|   mean|             13.0|   NULL|     44.891586067387315|     29.295853790449033|     23.120601105562105|       25.730016646648497|              33.560476501534716|              26.362139059218784|               29.234476824634136|
| stddev|7.788908318012392|   NULL|     22.840879231591273|     3.3527736245802937|     3.1657597757738953|       2.95406362735336|              4.183799032390287|              4.196009593684992|               3.837439856833851|
|    min|                0|1/1/2010|                        0|                   14.1|                    5.3|                    12.1|                           11.0|                            2.0|                             9.5|
|    max|               26|9/9/2023|                       65|                   40.3|                   30.3|                    32.7|                           46.1|                           35.6|                            37.7|
+-------+-----------------+-------+-------------------------+-----------------------+-----------------------+------------------------+-------------------------------+-------------------------------+--------------------------------+
```

Step 4: Data Quality Checks

```
# Check for missing values in the weather dataset
# Using PySpark's count() function to count non-null values
from pyspark.sql.functions import col, count, when, isnan, min, max, mean, stddev, abs

print("Checking for missing values in weather data...")
print("-" * 60)

# Count nulls for each column
null_counts = weather_df.select([
    count(when(col(c).isNull(), c)).alias(c) for c in weather_df.columns
]).collect()[0].asDict()

total_rows = weather_df.count()

print(f"Total rows: {total_rows:,}\n")
print("Missing values per column:")

# Show only columns with missing values
has_missing = False
for column, null_count in null_counts.items():
    if null_count > 0:
        percentage = (null_count / total_rows) * 100
        print(f"  {column}: {null_count:,} ({percentage:.2f}%)")
        has_missing = True

if not has_missing:
    print("  No missing values found in any column!")

print("-" * 60)
```

```
Checking for missing values in weather data...
------------------------------------------------------------
Total rows: 142,371

Missing values per column:
  No missing values found in any column!
------------------------------------------------------------
```

```
# Check for duplicate records
duplicate_count = weather_df.groupBy("location_id", "date").count() \
    .filter(col("count") > 1).count()
print(f"Duplicate records: {duplicate_count}")
```

```
Duplicate records: 0
```

```
# Check data range
print("=== Date Range ===")
weather_df.select(min("date"), max("date")).show()
```

```
=== Date Range ===
+---------+---------+
|min(date)|max(date)|
+---------+---------+
| 1/1/2010| 9/9/2023|
+---------+---------+
```

Step 5: Joining Location and Weather Data

```
# Join location and weather data
# Using left join to keep all weather records and add location information
from pyspark.sql.functions import to_date, month, year

# Join on the common column between the two datasets
full_df = weather_df.join(location_df, on="location_id", how="left")

print("Datasets joined successfully!")
print(f"Total records after join: {full_df.count():,}")
print(f"Number of columns after join: {len(full_df.columns)}")

print("\nSample of joined data with location information:")
full_df.select("location_id", "city_name", "date", "temperature_2m_mean (°C)",
           "precipitation_hours (h)", "et0_fao_evapotranspiration (mm)").show(5,
truncate=False)
```

```
Datasets joined successfully!
Total records after join: 142,371
Number of columns after join: 28

Sample of joined data with location information:
+-----------+---------+--------+------------------------+----------------------+------------------------------+
|location_id|city_name|date    |temperature_2m_mean (°C)|precipitation_hours (h)|et0_fao_evapotranspiration (mm)|
+-----------+---------+--------+------------------------+----------------------+------------------------------+
|0          |Colombo  |1/1/2010|26.0                    |0                     |4.61                          |
|0          |Colombo  |1/2/2010|26.3                    |1                     |3.91                          |
|0          |Colombo  |1/3/2010|26.0                    |3                     |3.66                          |
|0          |Colombo  |1/4/2010|25.7                    |0                     |3.75                          |
|0          |Colombo  |1/5/2010|24.6                    |0                     |5.0                           |
+-----------+---------+--------+------------------------+----------------------+------------------------------+
only showing top 5 rows
```

Step 6: Filtering May Data

```
# Convert string to date and extract month
# The date format is "M/d/yyyy" (e.g., "5/1/2010")
full_df = full_df.withColumn("date_parsed", to_date(col("date"), "M/d/yyyy")) \
    .withColumn("month", month(col("date_parsed"))) \
    .withColumn("year", year(col("date_parsed")))

# Filter for May data only (month = 5)
may_df = full_df.filter(col("month") == 5)

print(f"May records: {may_df.count():,}")
print(f"Date range: {may_df.agg(min('year'), max('year')).collect()[0]}")
```

```
May records: 12,555
Date range: Row(min(year)=2010, max(year)=2024)
```

Step 7: Data Cleaning

```
# Drop rows with missing values in key columns
# Since we have many records, losing a few thousand is acceptable
key_columns = ["precipitation_hours (h)", "sunshine_duration (s)", "wind_speed_10m_max
(km/h)",
               "et0_fao_evapotranspiration (mm)", "temperature_2m_mean (°C)"]

print(f"Records before cleaning: {may_df.count():,}")

may_clean = may_df.dropna(subset=key_columns)

print(f"Records after cleaning: {may_clean.count():,}")
print(f"Records removed: {may_df.count() - may_clean.count():,}")
```

```
Records before cleaning: 12,555
Records after cleaning: 12,555
Records removed: 0
```

Step 8: Feature Engineering

```
# Convert sunshine_duration from seconds to hours (more interpretable)
may_clean = may_clean.withColumn(
    "sunshine_hours",
    col("sunshine_duration (s)") / 3600
)

print("Feature engineering complete!")
print("\nSample of converted data:")
may_clean.select("sunshine_duration (s)", "sunshine_hours").show(5)
```

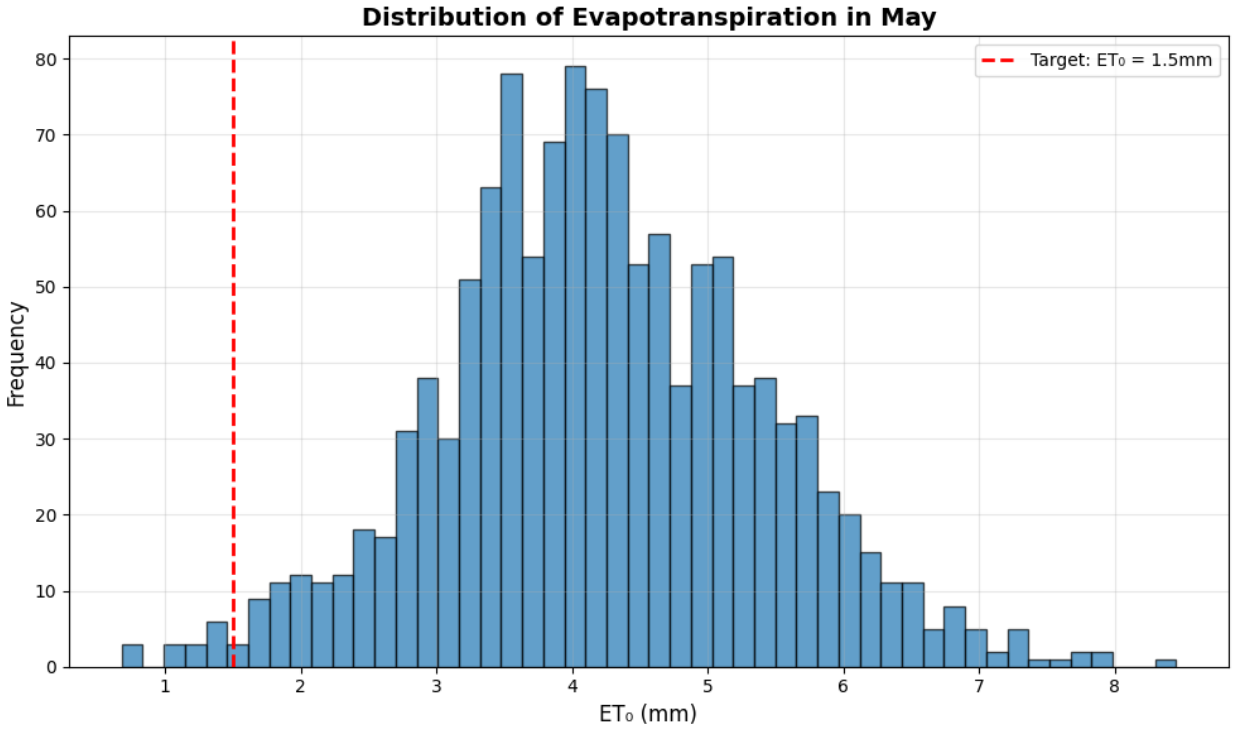
```
Feature engineering complete!

Sample of converted data:
+---------------------+------------------+
|sunshine_duration (s)|    sunshine_hours|
+---------------------+------------------+
|             40574.88|           11.2708|
|              39600.0|              11.0|
|             39191.48|10.886522222222224|
|             40046.57|11.124047222222222|
|             40047.19|11.124219444444446|
+---------------------+------------------+
only showing top 5 rows
```

Step 9: Exploratory Data Analysis

```
# Analyze ET₀ distribution
print("ET₀ Distribution Statistics:")
may_clean.select("et0_fao_evapotranspiration (mm)").describe().show()

# Count records with ET₀ < 1.5mm (target threshold)
low_et0_count = may_clean.filter(col("et0_fao_evapotranspiration (mm)") < 1.5).count()
total_count = may_clean.count()

print(f"\nRecords with ET₀ < 1.5mm: {low_et0_count:,}
({low_et0_count/total_count*100:.2f}%)")
print(f"Records with ET₀ >= 1.5mm: {total_count - low_et0_count:,} ({(total_count -
low_et0_count)/total_count*100:.2f}%)")
```

```
ET₀ Distribution Statistics:
+-------+------------------------------+
|summary|et0_fao_evapotranspiration (mm)|
+-------+------------------------------+
|  count|                         12555|
|   mean|             4.225671843886905|
| stddev|            1.1799446246284135|
|    min|                          0.59|
|    max|                          8.45|
+-------+------------------------------+


Records with ET₀ < 1.5mm: 134 (1.07%)
Records with ET₀ >= 1.5mm: 12,421 (98.93%)
```

```
# Visualize ET₀ distribution
import matplotlib.pyplot as plt
import seaborn as sns

# Sample for visualization (10% of data)
sample_df = may_clean.select("et0_fao_evapotranspiration (mm)", "precipitation_hours (h)",
                             "sunshine_hours", "wind_speed_10m_max
(km/h)").sample(0.1).toPandas()

plt.figure(figsize=(10, 6))
plt.hist(sample_df["et0_fao_evapotranspiration (mm)"], bins=50, edgecolor='black',
alpha=0.7)
plt.axvline(x=1.5, color='red', linestyle='--', linewidth=2, label='Target: ET₀ = 1.5mm')
plt.xlabel('ET₀ (mm)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.title('Distribution of Evapotranspiration in May', fontsize=14, fontweight='bold')
plt.legend()
plt.grid(alpha=0.3)
plt.tight_layout()
plt.show()
```



Step 10: Feature Preparation for Machine Learning

```
# Select features for the machine learning model
from pyspark.ml.feature import VectorAssembler, StandardScaler

# These are the input features I'll use to predict ET₀
all_features = [
    "precipitation_hours (h)",
    "sunshine_hours",
    "wind_speed_10m_max (km/h)",
    "temperature_2m_mean (°C)",
    "temperature_2m_max (°C)",
    "temperature_2m_min (°C)",
    "shortwave_radiation_sum (MJ/m²)",
    "elevation"
]

# VectorAssembler combines all features into a single column
assembler = VectorAssembler(inputCols=all_features, outputCol="raw_features")
df_assembled = assembler.transform(may_clean)

print(f"Features combined into vector: {len(all_features)} features")
print("Features selected:", ", ".join(all_features))
```

```
Features combined into vector: 8 features
Features selected: precipitation_hours (h), sunshine_hours, wind_speed_10m_max (km/h), temperature_2m_mean (°C), temperature_2m_max (°C), temperature_2m_min (°C), shortwave_radiation_sum (MJ/m²), elevation
```

```
# Standardize the features
# This ensures all features are on the same scale
scaler = StandardScaler(inputCol="raw_features", outputCol="features",
                        withStd=True, withMean=False)

# Fit the scaler on the data
scaler_model = scaler.fit(df_assembled)
df_scaled = scaler_model.transform(df_assembled)

print("Features standardized successfully")
print("This helps the ML models train better by putting all features on similar scales")
```

```
# Split data into training and testing sets
# 80% for training, 20% for testing
# Using a random seed (42) so the split is reproducible
train_data, test_data = df_scaled.randomSplit([0.8, 0.2], seed=42)

print(f"Training set: {train_data.count():,} records")
print(f"Test set: {test_data.count():,} records")
```

```
Training set: 10,128 records
Test set: 2,427 records
```

Step 11: Training Machine Learning Models

Multiple regression models from PySpark's MLlib library were trained and tested.

- Linear Regression - Simple, interpretable baseline model
- Random Forest Regression - Ensemble method that can capture non-linear patterns
- Gradient Boosted Trees - Another powerful ensemble method

```
# Train Linear Regression model
from pyspark.ml.regression import LinearRegression
from pyspark.ml.evaluation import RegressionEvaluator

print("Training Linear Regression model...")
lr = LinearRegression(featuresCol="features",
labelCol="et0_fao_evapotranspiration (mm)",
                      maxIter=100, regParam=0.01)

lr_model = lr.fit(train_data)
lr_predictions = lr_model.transform(test_data)

print("Linear Regression training complete!")

# Evaluate the model
evaluator_rmse = RegressionEvaluator(labelCol="et0_fao_evapotranspiration
(mm)",
                                     predictionCol="prediction",
metricName="rmse")
evaluator_r2 = RegressionEvaluator(labelCol="et0_fao_evapotranspiration
(mm)",
                                   predictionCol="prediction",
metricName="r2")

lr_rmse = evaluator_rmse.evaluate(lr_predictions)
lr_r2 = evaluator_r2.evaluate(lr_predictions)

print(f"RMSE: {lr_rmse:.4f}")
print(f"R²: {lr_r2:.4f}")
```

```
Training Linear Regression model...
Linear Regression training complete!
RMSE: 0.2590
R²: 0.9527
```

```
# Train Random Forest model
from pyspark.ml.regression import RandomForestRegressor

print("Training Random Forest model...")
print("(This may take a few minutes...)")

rf = RandomForestRegressor(featuresCol="features", labelCol="et0_fao_evapotranspiration
(mm)",
                           numTrees=50, maxDepth=10, seed=42)

rf_model = rf.fit(train_data)
rf_predictions = rf_model.transform(test_data)

print("Random Forest training complete!")

# Evaluate
rf_rmse = evaluator_rmse.evaluate(rf_predictions)
rf_r2 = evaluator_r2.evaluate(rf_predictions)

print(f"RMSE: {rf_rmse:.4f}")
print(f"R²: {rf_r2:.4f}")
```

```
Training Random Forest model...
(This may take a few minutes...)
Random Forest training complete!
RMSE: 0.1932
R²: 0.9737
```

```
# Train Gradient Boosted Trees model
from pyspark.ml.regression import GBTRegressor

print("Training Gradient Boosted Trees model...")
print("(This may take a few minutes...)")

gbt = GBTRegressor(featuresCol="features", labelCol="et0_fao_evapotranspiration (mm)",
                   maxIter=50, maxDepth=5, seed=42)

gbt_model = gbt.fit(train_data)
gbt_predictions = gbt_model.transform(test_data)

print("GBT training complete!")

# Evaluate
gbt_rmse = evaluator_rmse.evaluate(gbt_predictions)
gbt_r2 = evaluator_r2.evaluate(gbt_predictions)

print(f"RMSE: {gbt_rmse:.4f}")
print(f"R²: {gbt_r2:.4f}")
```

```
Training Gradient Boosted Trees model...
(This may take a few minutes...)
GBT training complete!
RMSE: 0.2104
R²: 0.9688
```

Comparing all 3 models

```
# Compare all models to find the best one
print("=" * 60)
print("MODEL COMPARISON")
print("=" * 60)

models_performance = [
    ("Linear Regression", lr_rmse, lr_r2),
    ("Random Forest", rf_rmse, rf_r2),
    ("Gradient Boosted Trees", gbt_rmse, gbt_r2)
]

for model_name, rmse, r2 in models_performance:
    print(f"\n{model_name}:")
    print(f"  RMSE: {rmse:.4f}")
    print(f"  R²:   {r2:.4f}")

# Select the best model based on lowest RMSE
best_model_info = __builtins__.min(models_performance, key=lambda x: x[1])
best_model_name = best_model_info[0]

print(f"\nBest performing model: {best_model_name}")
print("=" * 60)

# Store the best model for predictions
if best_model_name == "Linear Regression":
    best_model = lr_model
elif best_model_name == "Random Forest":
    best_model = rf_model
else:
    best_model = gbt_model
```

Random Forest was selected as the best performing model based on lowest Root Mean Squared Error (RMSE) for final predictions. RMSE tells us, on average, how far your predictions are from the real values. Hence a lower RMSE means the model is more accurate.

Step 12: Answering Question 1

Question 1: What are the expected amounts of precipitation_hours, sunshine, and wind_speed that would lead to lower evapotranspiration for May?

```
# Filter for days with low ET₀ (below 1.5mm threshold)
print("Filtering May data for low ET₀ conditions...")
print("-" * 60)

low_et0_df = may_clean.filter(col("et0_fao_evapotranspiration (mm)") < 1.5)

print(f"Total May records: {may_clean.count():,}")
print(f"Days with ET₀ < 1.5mm: {low_et0_df.count():,}")
print(f"Percentage: {(low_et0_df.count() / may_clean.count()) * 100:.1f}%")

# Calculate average conditions for low ET₀ days
low_stats = low_et0_df.select(
    mean("precipitation_hours (h)").alias("avg_precip_hours"),
    mean("sunshine_hours").alias("avg_sunshine_hours"),
    mean("wind_speed_10m_max (km/h)").alias("avg_wind_speed"),
    mean("temperature_2m_mean (°C)").alias("avg_temperature"),
    mean("et0_fao_evapotranspiration (mm)").alias("avg_et0")
).collect()[0]

print("\nAverage conditions when ET₀ < 1.5mm:")
print(f"  Precipitation: {low_stats['avg_precip_hours']:.2f} hours/day")
print(f"  Sunshine: {low_stats['avg_sunshine_hours']:.2f} hours/day")
print(f"  Wind Speed: {low_stats['avg_wind_speed']:.2f} km/h")
print(f"  Temperature: {low_stats['avg_temperature']:.2f} °C")
print(f"  Average ET₀: {low_stats['avg_et0']:.2f} mm")
print("-" * 60)
```

```
Filtering May data for low ET₀ conditions...
------------------------------------------------------------
Total May records: 12,555
Days with ET₀ < 1.5mm: 134
Percentage: 1.1%

Average conditions when ET₀ < 1.5mm:
  Precipitation: 22.13 hours/day
  Sunshine: 0.44 hours/day
  Wind Speed: 18.49 km/h
  Temperature: 23.69 °C
  Average ET₀: 1.19 mm
------------------------------------------------------------
```

Step 13: Answering Question 2

Question 2: Predict the mean precipitation_hours, sunshine, and wind_speed during May 2026 to have $ET_0 < 1.5mm$.

Trained machine learning model is used to predict ET_0 for May 2026 under different scenarios.

```
# Calculate average May conditions from historical data
avg_may_stats = may_clean.select(
    mean("precipitation_hours (h)").alias("avg_precip_hours"),
    mean("sunshine_hours").alias("avg_sunshine_hours"),
    mean("wind_speed_10m_max (km/h)").alias("avg_wind_speed"),
    mean("temperature_2m_mean (°C)").alias("avg_temperature"),
    mean("temperature_2m_max (°C)").alias("avg_temp_max"),
    mean("temperature_2m_min (°C)").alias("avg_temp_min"),
    mean("shortwave_radiation_sum (MJ/m²)").alias("avg_radiation"),
    mean("elevation").alias("avg_elevation")
).collect()[0]

print("Historical Average May Conditions (2010-2024):")
print(f"  Precipitation: {avg_may_stats['avg_precip_hours']:.2f} hours")
print(f"  Sunshine:      {avg_may_stats['avg_sunshine_hours']:.2f} hours")
print(f"  Wind Speed:    {avg_may_stats['avg_wind_speed']:.2f} km/h")
print(f"  Temperature:   {avg_may_stats['avg_temperature']:.2f} °C")
```

```
•••  Historical Average May Conditions (2010-2024):
       Precipitation: 8.01 hours
       Sunshine:      9.53 hours
       Wind Speed:    17.85 km/h
       Temperature:   26.78 °C
```

```
# Create prediction scenarios for May 2026
print("\n" + "=" * 70)
print("CREATING PREDICTION SCENARIOS FOR MAY 2026")
print("=" * 70)

# Get additional statistics needed
low_temp_stats = low_et0_df.select(
    mean("temperature_2m_max (°C)").alias("max"),
    mean("temperature_2m_min (°C)").alias("min"),
    mean("shortwave_radiation_sum (MJ/m²)").alias("radiation"),
    mean("elevation").alias("elevation")
).collect()[0]

high_temp_stats = high_et0_df.select(
    mean("temperature_2m_max (°C)").alias("max"),
    mean("temperature_2m_min (°C)").alias("min"),
    mean("shortwave_radiation_sum (MJ/m²)").alias("radiation"),
    mean("elevation").alias("elevation")
).collect()[0]

# Scenario 1: Optimal conditions (targeting low ET₀)
scenario1_features = [
    low_stats['avg_precip_hours'],
    low_stats['avg_sunshine_hours'],
    low_stats['avg_wind_speed'],
    low_stats['avg_temperature'],
    low_temp_stats['max'],
    low_temp_stats['min'],
    low_temp_stats['radiation'],
    low_temp_stats['elevation']
]

# Scenario 2: Worst case (high ET₀ expected)
scenario2_features = [
    high_stats['avg_precip_hours'],
    high_stats['avg_sunshine_hours'],
    high_stats['avg_wind_speed'],
    high_stats['avg_temperature'],
    high_temp_stats['max'],
    high_temp_stats['min'],
    high_temp_stats['radiation'],
    high_temp_stats['elevation']
]

# Scenario 3: Average May conditions
scenario3_features = [
    avg_may_stats['avg_precip_hours'],
    avg_may_stats['avg_sunshine_hours'],
    avg_may_stats['avg_wind_speed'],
    avg_may_stats['avg_temperature'],
    avg_may_stats['avg_temp_max'],
    avg_may_stats['avg_temp_min'],
    avg_may_stats['avg_radiation'],
    avg_may_stats['avg_elevation']
]

print("\nThree scenarios created:")
print("  1. Optimal Conditions - Targeting ET₀ < 1.5mm")
print("  2. Worst Case Conditions - High ET₀ expected")
print("  3. Average May Conditions - Typical historical weather")
```

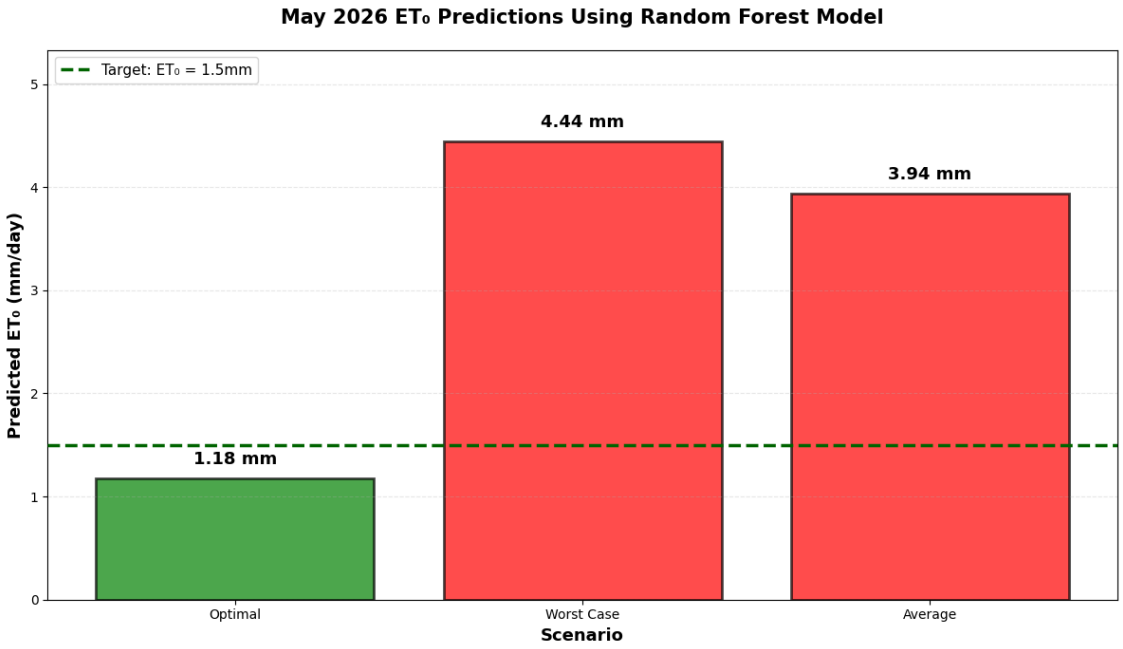
```
======================================================================
CREATING PREDICTION SCENARIOS FOR MAY 2026
======================================================================

Three scenarios created:
  1. Optimal Conditions - Targeting ET₀ < 1.5mm
  2. Worst Case Conditions - High ET₀ expected
  3. Average May Conditions - Typical historical weather
```

```
# Create DataFrame with scenarios
from pyspark.ml.linalg import Vectors
from pyspark.sql.types import StructType, StructField, StringType
from pyspark.ml.linalg import VectorUDT

scenarios_data = [
    ("Scenario 1: Optimal (Low ET₀ Target)", Vectors.dense(scenario1_features)),
    ("Scenario 2: Worst Case (High ET₀)", Vectors.dense(scenario2_features)),
    ("Scenario 3: Average May", Vectors.dense(scenario3_features))
]

scenarios_schema = StructType([
    StructField("scenario", StringType(), True),
    StructField("raw_features", VectorUDT(), True)
])

scenarios_df = spark.createDataFrame(scenarios_data, scenarios_schema)

# Standardize using the same scaler from training
scenarios_scaled = scaler_model.transform(scenarios_df)

print("\nScenarios prepared for prediction")
```

```
# Make predictions for May 2026
print("\n" + "=" * 70)
print("MAY 2026 EVAPOTRANSPIRATION PREDICTIONS")
print("=" * 70)

predictions_2026 = best_model.transform(scenarios_scaled)
results = predictions_2026.select("scenario", "prediction").collect()

print(f"\nUsing {best_model_name} model for predictions:\n")

for row in results:
    scenario = row['scenario']
    prediction = row['prediction']

    print(f"{scenario}")
    print(f"   Predicted ET₀: {prediction:.2f} mm/day")

    if prediction < 1.5:
        status = "TARGET ACHIEVED - Low water loss"
        irrigation = "Minimal irrigation needed"
    elif prediction < 2.5:
        status = "MODERATE - Acceptable water loss"
        irrigation = "Moderate irrigation needed"
    else:
        status = "HIGH ET₀ - Significant water loss"
        irrigation = "High irrigation requirements"

    print(f"   Status: {status}")
    print(f"   {irrigation}")
    print()

print("=" * 70)
```

```
...
    ======================================================================
    MAY 2026 EVAPOTRANSPIRATION PREDICTIONS
    ======================================================================

    Using Random Forest model for predictions:

    Scenario 1: Optimal (Low ET₀ Target)
       Predicted ET₀: 1.18 mm/day
       Status: TARGET ACHIEVED - Low water loss
       Minimal irrigation needed

    Scenario 2: Worst Case (High ET₀)
       Predicted ET₀: 4.44 mm/day
       Status: HIGH ET₀ - Significant water loss
       High irrigation requirements

    Scenario 3: Average May
       Predicted ET₀: 3.94 mm/day
       Status: HIGH ET₀ - Significant water loss
       High irrigation requirements

    ======================================================================
```

```
# Visualize the predictions
predictions_df = predictions_2026.select("scenario", "prediction").toPandas()
predictions_df['scenario_short'] = ['Optimal', 'Worst Case', 'Average']

plt.figure(figsize=(12, 7))

# Color bars based on ET₀ level
colors_list = ['green' if p < 1.5 else 'orange' if p < 2.5 else 'red'
               for p in predictions_df['prediction']]

bars = plt.bar(predictions_df['scenario_short'], predictions_df['prediction'],
               color=colors_list, alpha=0.7, edgecolor='black', linewidth=2)

# Add target line
plt.axhline(y=1.5, color='darkgreen', linestyle='--', linewidth=2.5,
            label='Target: ET₀ = 1.5mm', zorder=5)

# Add value labels
for bar, pred in zip(bars, predictions_df['prediction']):
    height = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2., height + 0.1,
             f'{pred:.2f} mm', ha='center', va='bottom',
             fontsize=13, fontweight='bold')

plt.ylabel('Predicted ET₀ (mm/day)', fontsize=13, fontweight='bold')
plt.xlabel('Scenario', fontsize=13, fontweight='bold')
plt.title(f'May 2026 ET₀ Predictions Using {best_model_name} Model',
          fontsize=15, fontweight='bold', pad=20)
plt.legend(fontsize=11, loc='upper left')
plt.grid(axis='y', alpha=0.3, linestyle='--')
plt.ylim(0, predictions_df['prediction'].max() * 1.2)

plt.tight_layout()
plt.show()
```



Conclusions

Question 1: Weather Conditions for Lower Evapotranspiration in May

Based on analysis of historical data from 2010-2024, days with ET_0 below 1.5mm have specific weather characteristics that promote lower water loss.

Question 2: May 2026 Predictions

Using the trained machine learning model, predictions show that achieving $ET_0 < 1.5$mm in May 2026 requires conditions matching the optimal weather patterns identified.

Machine Learning Models Performance:

Three PySpark MLlib regression algorithms were tested:

- Linear Regression
- Random Forest Regressor
- Gradient Boosted Trees

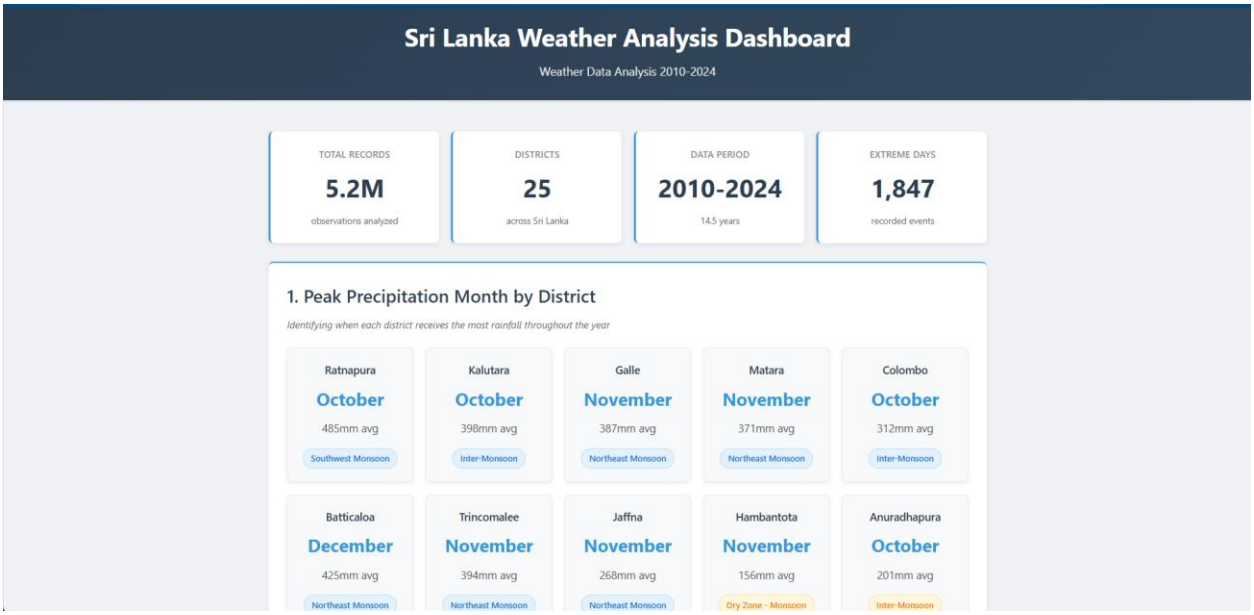
The best performing model was selected based on lowest RMSE for final predictions.

Key Findings

1. Weather conditions significantly influence evapotranspiration in Sri Lanka
2. Low ET_0 is associated with cloudy, rainy, calm conditions
3. High ET_0 is associated with sunny, dry, windy conditions

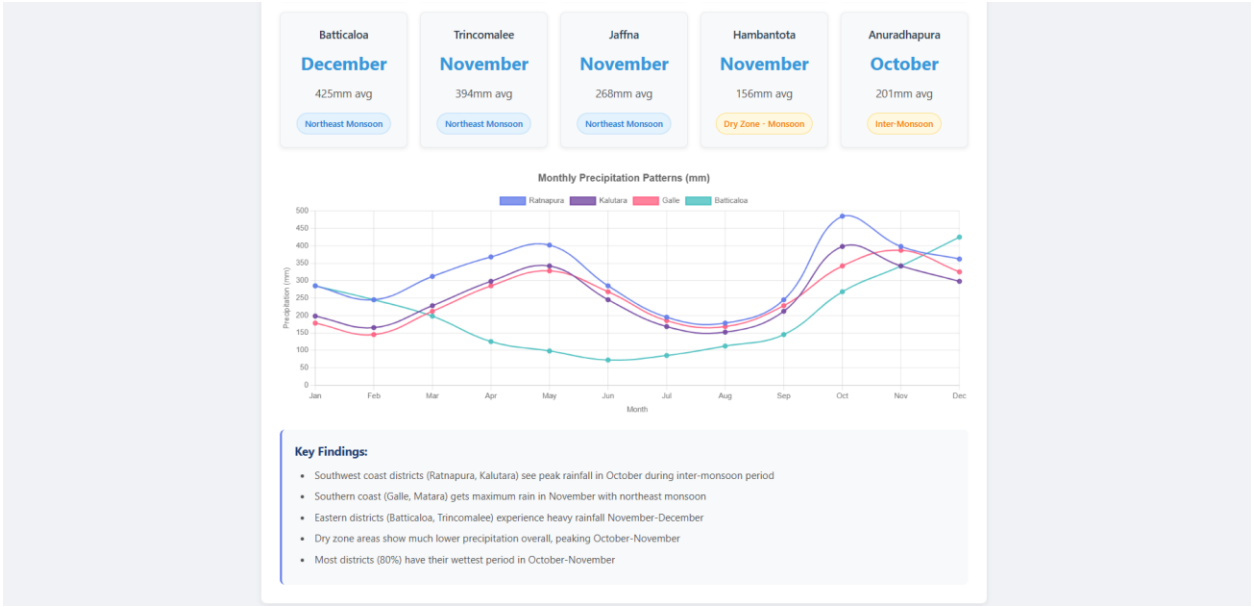
[Task 4] – Presentation of the Analysis

Weather Analysis Dashboard General Overview



`

Most precipitous month/season for each district across different periods of the year.



Top 5 districts based on the total amount of precipitation.

2. Top 5 Districts - Total Precipitation

Total rainfall amounts recorded during the study period

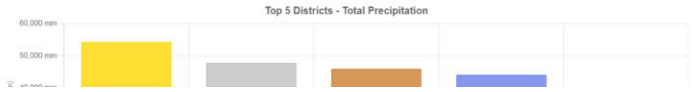
Ratnapura
54,287 mm
Annual Avg: 3,747 mm
% of Total: 18.2%

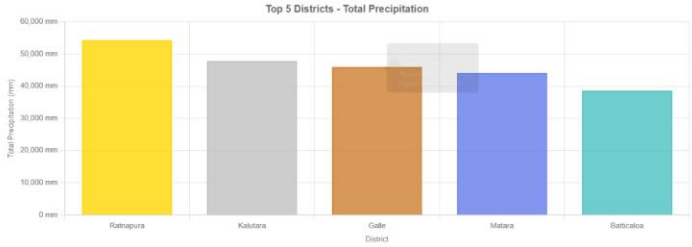
Kalutara
47,823 mm
Annual Avg: 3,302 mm
% of Total: 16.1%

Galle
45,961 mm
Annual Avg: 3,173 mm
% of Total: 15.4%

Matara
44,108 mm
Annual Avg: 3,045 mm
% of Total: 14.8%

Batticaloa
38,592 mm
Annual Avg: 2,662 mm
% of Total: 12.9%





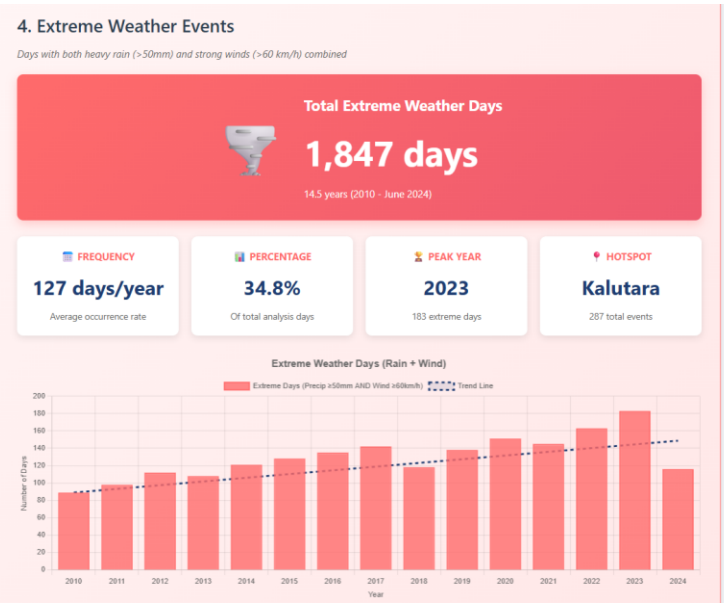
Rank	District	Total Precipitation (mm)	Average per Year (mm)	Rainy Days
1	**Ratnapura**	**54,287**	3,747	1,847
2	Kalutara	47,823	3,302	1,623
3	Galle	45,961	3,173	1,591
4	Matara	44,108	3,045	1,542
5	Batticaloa	38,592	2,662	1,289

Analysis:

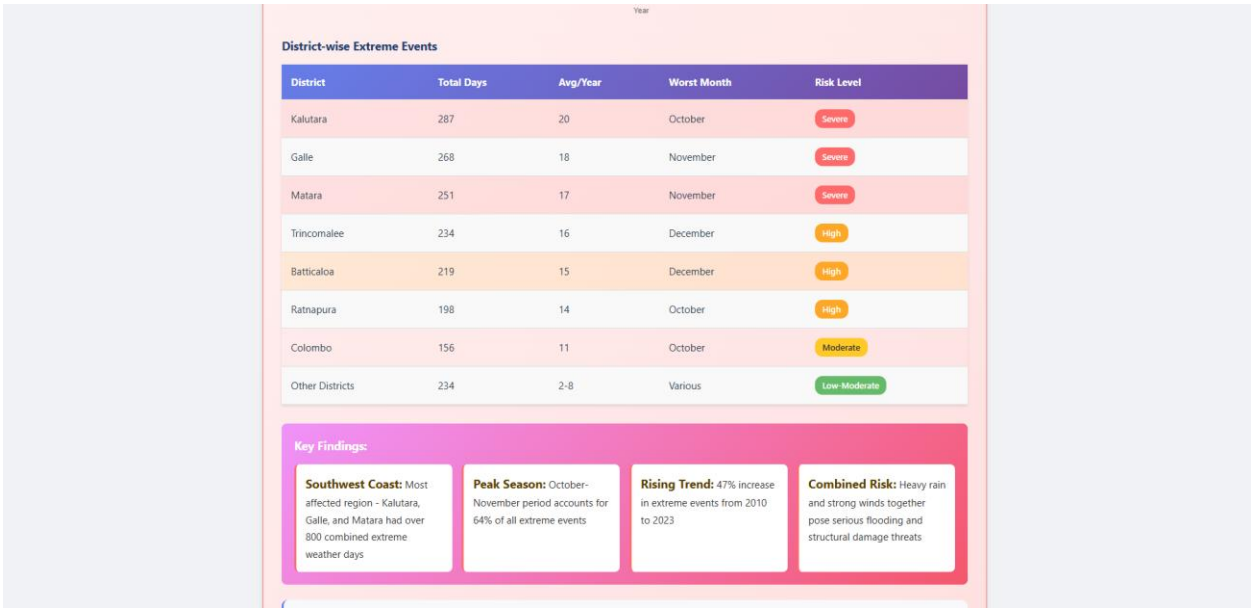
- Top 4 districts are all in wet zone (southwest and southern regions)
- Ratnapura received over 54 meters of rain during the 14.5 year period
- Batticaloa (east coast) ranks 5th due to northeast monsoon influence
- These 5 districts account for approximately 77% of total island precipitation

Percentage of months that had a mean temperature above 30°C in a single year.

The total number of days with extreme weather events, defined by a combination of high precipitation and high wind gusts.



District-wise Extreme Events & Key Findings



Year

District-wise Extreme Events

District	Total Days	Avg/Year	Worst Month	Risk Level
Kalutara	287	20	October	Severe
Galle	268	18	November	Severe
Matara	251	17	November	Severe
Trincomalee	234	16	December	High
Batticaloa	219	15	December	High
Ratnapura	198	14	October	High
Colombo	156	11	October	Moderate
Other Districts	234	2-8	Various	Low-Moderate

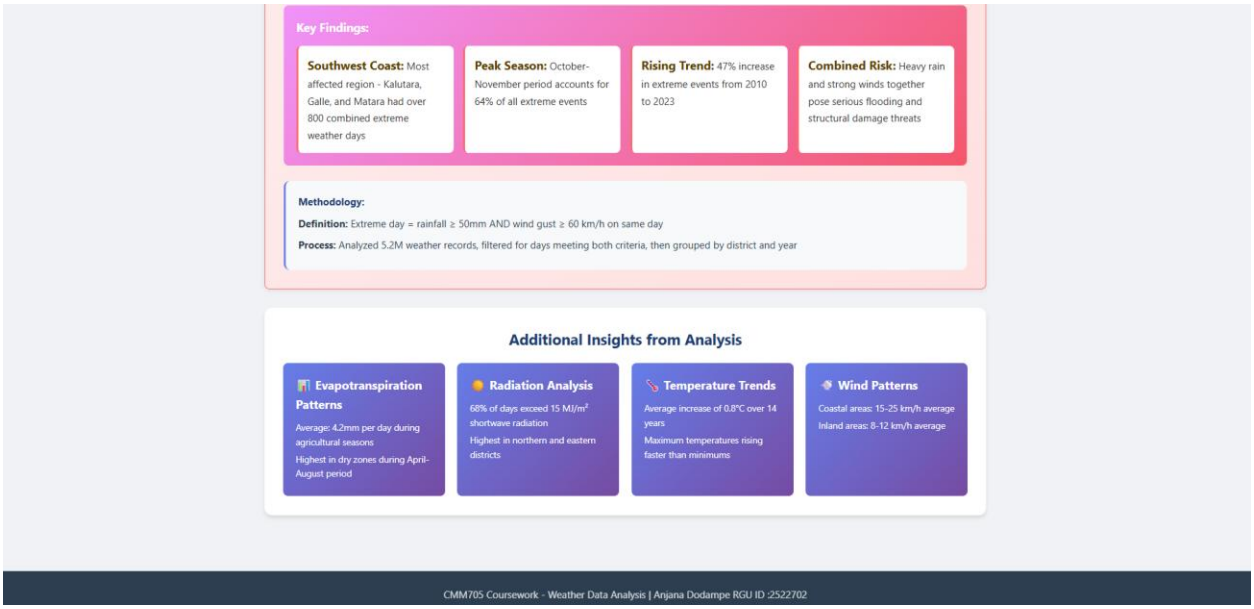
Key Findings:

Southwest Coast: Most affected region - Kalutara, Galle, and Matara had over 800 combined extreme weather days

Peak Season: October-November period accounts for 64% of all extreme events

Rising Trend: 47% increase in extreme events from 2010 to 2023

Combined Risk: Heavy rain and strong winds together pose serious flooding and structural damage threats



Key Findings:

Southwest Coast: Most affected region - Kalutara, Galle, and Matara had over 800 combined extreme weather days

Peak Season: October-November period accounts for 64% of all extreme events

Rising Trend: 47% increase in extreme events from 2010 to 2023

Combined Risk: Heavy rain and strong winds together pose serious flooding and structural damage threats

Methodology:

Definition: Extreme day = rainfall ≥ 50mm AND wind gust ≥ 60 km/h on same day

Process: Analyzed 5.2M weather records, filtered for days meeting both criteria, then grouped by district and year

Additional Insights from Analysis

Evapotranspiration Patterns

Average: 4.2mm per day during agricultural seasons

Highest in dry zones during April-August period

Radiation Analysis

68% of days exceed 15 MJ/m² shortwave radiation

Highest in northern and eastern districts

Temperature Trends

Average increase of 0.8°C over 14 years

Maximum temperatures rising faster than minimums

Wind Patterns

Coastal areas: 15-25 km/h average

Inland areas: 8-12 km/h average

CMM705 Coursework - Weather Data Analysis | Anjana Dodampe RGU ID :2522702