cat weatherHistory.csv | python mapper.py | sort | python reducer.py

**1. 📄 Dataset (Input File)**

You are working with a file like weatherHistory.csv, which contains **weather observations**.

Sample:

mathematica

CopyEdit

Formatted Date,Summary,Precip Type,Temperature (C),Apparent Temperature (C),Humidity,...

2006-04-01 00:00:00+00:00,Partly Cloudy,rain,9.4722,7.3887,0.89,...

2006-04-01 01:00:00+00:00,Partly Cloudy,rain,9.3556,7.2272,0.86,...

...

From this dataset:

* **Year** comes from the **Formatted Date**.
* **Temperature** comes from the **Temperature (C)**.

**2. 🛠 Goal**

Using **MapReduce**, you need to **find**:

* Which **year** had the **hottest recorded temperature**.
* Which **year** had the **coolest recorded temperature**.

**3. ⚙️ How the Code Works (Step-by-Step)**

**📍 Step 1: mapper.py**

**Purpose:**

* Read each row.
* Extract **Year** and **Temperature**.
* Emit (print) key-value pairs like:  
  Year Temperature

**✏️ Code Walkthrough: mapper.py**

python

CopyEdit

import sys

for line in sys.stdin: # Read input line-by-line from standard input

parts = line.strip().split(",") # Remove extra spaces and split by commas

if parts[0] == "Formatted Date":

continue # Skip the first line (column headers)

try:

date = parts[0] # Date is the first column

temperature = float(parts[3]) # Temperature (C) is the fourth column

year = date.split("-")[0] # Split 'YYYY-MM-DD' and take 'YYYY'

print(f"{year}\t{temperature}") # Print key-value separated by TAB

except:

continue # Skip lines that cause error (bad data)

**What mapper outputs:**  
Example:

yaml

CopyEdit

2006 9.4722

2006 9.3556

2006 8.2889

2007 14.3556

2008 22.1778

...

**⚡ Important Points:**

* **sys.stdin** allows reading **piped data** (from cat weatherHistory.csv).
* **.split(",")** assumes the file is **comma-separated** (CSV).
* **Try-Except** ensures no crash if one row has missing values.

**📍 Step 2: sort**

**Purpose:**  
Sorts all mapper outputs based on **Year** first, then Temperature second.

Command:

bash

CopyEdit

sort

It arranges the output alphabetically, which makes **grouping** easy for the reducer.

**📍 Step 3: reducer.py**

**Purpose:**

* Read sorted mapper output.
* Find:
  + Maximum Temperature (hottest)
  + Minimum Temperature (coolest)
* Track the corresponding **Year** for each.

**✏️ Code Walkthrough: reducer.py**

python

CopyEdit

import sys

hottest\_year = None

coolest\_year = None

max\_temp = float('-inf') # Start with very low number

min\_temp = float('inf') # Start with very high number

for line in sys.stdin:

year, temp = line.strip().split("\t") # Split Year and Temperature

temp = float(temp) # Convert Temperature to number

if temp > max\_temp:

max\_temp = temp

hottest\_year = year

if temp < min\_temp:

min\_temp = temp

coolest\_year = year

# Finally, print results

print(f"Hottest Year: {hottest\_year} with Temperature: {max\_temp}°C")

print(f"Coolest Year: {coolest\_year} with Temperature: {min\_temp}°C")

**What reducer does internally:**

* Keeps comparing **each temperature**.
* Updates:
  + **hottest\_year** if a **higher temperature** found.
  + **coolest\_year** if a **lower temperature** found.

**🔥 Output Example**

After the entire pipeline:

sql

CopyEdit

Hottest Year: 2015 with Temperature: 38.22°C

Coolest Year: 2007 with Temperature: -17.78°C

**4. 🖇️ Overall Data Flow Diagram**

scss

CopyEdit

Input (weatherHistory.csv)

↓

Mapper (mapper.py) → Emits (Year, Temperature)

↓

Sort (sort)

↓

Reducer (reducer.py) → Calculates (hottest year, coolest year)

↓

Final Output

**5. ⚡ Why MapReduce here?**

| **Feature** | **Why Useful** |
| --- | --- |
| **Scalability** | Handles very large weather datasets easily. |
| **Efficiency** | Splits work between many machines if needed. |
| **Fault tolerance** | Even if one small process fails, MapReduce retries it. |

This is the **core power** behind using **MapReduce** even for this simple problem.

**🎯 Final Key Points to Remember**

✅ **mapper.py** → Extracts (Year, Temperature)  
✅ **sort** → Groups similar years together  
✅ **reducer.py** → Finds max and min across all years  
✅ **Piping (|)** → Sends data from one program to another ✅ **Robust code** → Handles missing/invalid data using try-except

**What is MapReduce?**

* **MapReduce** is a **programming model** used for processing **large datasets** in a distributed system.
* It breaks the computation into two main steps:
  + **Map Step:** Process and transform input data into intermediate key-value pairs.
  + **Reduce Step:** Aggregate the intermediate data based on keys and produce final results.

✅ MapReduce is scalable, fault-tolerant, and works efficiently on massive data.

**3. 🎯 Aim of the Practical**

* Design and develop a distributed application using **MapReduce**.
* **Find:**
  + The **Hottest Year** (year with maximum recorded temperature).
  + The **Coolest Year** (year with minimum recorded temperature).
* **Dataset:** Weather data file (e.g., weatherHistory.csv).

**4. 🏗️ System Architecture**

| **Stage** | **Details** |
| --- | --- |
| **Input** | CSV file with weather records (Date, Temperature, etc.) |
| **Mapper** | Extracts (Year, Temperature) from each record |
| **Sort** | Groups data by Year |
| **Reducer** | Finds hottest and coolest year by comparing temperatures |
| **Output** | Displays the year with max and min temperature |

**5. ⚙️ Working of the System**

**📍 Mapper Phase**

* Reads each weather record.
* Extracts:
  + **Year** from the Date.
  + **Temperature** from the respective column.
* Emits:

scss

CopyEdit

(Year, Temperature)

**📍 Shuffle and Sort Phase**

* Sorts all emitted (Year, Temperature) pairs.
* Groups the same years together for easy reduction.

**📍 Reducer Phase**

* Reads sorted data.
* Compares temperatures:
  + Keeps track of the **maximum** and **minimum**.
* At the end:
  + Prints the Hottest Year and Coolest Year with corresponding temperatures.

**6. 🔥 Advantages of using MapReduce**

* **Handles Big Data** efficiently.
* **Parallelism**: Many mappers and reducers can run simultaneously.
* **Fault-tolerant**: Retries failed tasks automatically.
* **Cost-effective**: Works on clusters of inexpensive hardware.

**7. 🧠 Important Functions/Commands Used**

| **Function/Command** | **Purpose** |
| --- | --- |
| sys.stdin | Read standard input (piped data). |
| split(",") | Split CSV rows into columns. |
| print(f"{year}\t{temperature}") | Emit key-value pairs separated by Tab. |
| sort | Sort output from mapper based on Year and Temperature. |
| try-except | Handle bad data gracefully. |
| float('-inf') | Used to initialize minimum and maximum values. |

**8. 📈 Sample Output**

sql

CopyEdit

Hottest Year: 2015 with Temperature: 38.22°C

Coolest Year: 2007 with Temperature: -17.78°C

**9. 📋 Applications**

* Climate Change Analysis.
* Weather Forecasting Improvements.
* Agriculture Planning based on historical weather.
* Disaster Management (heat waves, cold waves studies).

**10. 📝 Conclusion**

In this practical, we designed a **distributed weather data analyzer** using **MapReduce**.  
The system reads large weather data, extracts important details, and finds the hottest and coolest years efficiently.

**Key Points:**

* MapReduce splits the job: Mapper (Extraction) → Reducer (Aggregation).
* The system can handle very large datasets.
* Distributed processing ensures faster results and better resource usage.