**SRM Institute of Science and Technology**

**Big Data Analytics with Applications**

**Lab 2: Develop a MapReduce program to calculate the frequency of a given word in a given file.**

**# Step 1: Create mapper.py**

with open("mapper.py", "w") as f:

f.write("""

#!/usr/bin/env python3

import sys

target\_word = "hadoop" # Change this to the word you want to count

for line in sys.stdin:

words = line.strip().lower().split()

for word in words:

if word == target\_word:

print(f"{word}\\t1")

""")

**# Step 2: Create reducer.py**

with open("reducer.py", "w") as f:

f.write("""

#!/usr/bin/env python3

import sys

current\_word = None

current\_count = 0

for line in sys.stdin:

word, count = line.strip().split('\\t')

count = int(count)

if current\_word == word:

current\_count += count

else:

if current\_word:

print(f"{current\_word}\\t{current\_count}")

current\_word = word

current\_count = count

if current\_word == word:

print(f"{current\_word}\\t{current\_count}")

""")

**# Step 3: Create input file**

with open("input.txt", "w") as f:

f.write("""Hadoop is an open-source framework. Hadoop allows for the distributed processing of large data sets across clusters of computers. Hadoop is reliable and scalable.""")

**# Step 4: Run the mapper and reducer (simulate Hadoop streaming)**

!cat input.txt | python3 mapper.py | sort | python3 reducer.py

**Lab 3: Develop a MapReduce program to find the maximum temperature in each year**

**# Step 1: Create mapper.py for extracting (year, temperature)**

with open("mapper.py", "w") as f:

f.write("""

#!/usr/bin/env python3

import sys

for line in sys.stdin:

line = line.strip()

if not line:

continue

try:

date, temp = line.split()

year = date[:4]

print(f"{year}\\t{temp}")

except ValueError:

continue

""")

**# Step 2: Create reducer.py to find max temperature per year**

with open("reducer.py", "w") as f:

f.write("""

#!/usr/bin/env python3

import sys

current\_year = None

max\_temp = float('-inf')

for line in sys.stdin:

year, temp = line.strip().split('\\t')

temp = int(temp)

if current\_year == year:

max\_temp = max(max\_temp, temp)

else:

if current\_year:

print(f"{current\_year}\\t{max\_temp}")

current\_year = year

max\_temp = temp

if current\_year:

print(f"{current\_year}\\t{max\_temp}")

""")

**# Step 3: Create input file (simulated weather data)**

with open("input.txt", "w") as f:

f.write("""1980-01-01 22

1980-05-11 38

1980-12-31 33

1981-04-04 41

1981-06-10 39

1982-03-09 29

1982-09-19 37

1982-11-22 35

""")

**# Step 4: Simulate Hadoop streaming (mapper | sort | reducer)**

!cat input.txt | python3 mapper.py | sort | python3 reducer.py

**Lab 4:Develop a MapReduce program to find the grades of student’s.**

**# Create a student\_marks.txt file with header and data**

with open('student\_marks.txt', 'w') as f:

    f.write('reg.no eng tam core allied\n')  # header line

    f.write('101 85 92 78 90 88\n')

    f.write('102 70 75 80 65 60\n')

    f.write('103 55 60 58 62 59\n')

**Main Program:**

def mapper(filename):

    student\_averages = {}

    with open(filename, 'r') as file:

        header = next(file)  # Skip the header line

        for line in file:

            parts = line.strip().split()

            reg\_no = parts[0]

            marks = list(map(int, parts[1:]))

            average = sum(marks) / len(marks)

            student\_averages[reg\_no] = average

    return student\_averages

def reducer(student\_averages):

    grades = {}

    for reg\_no, avg in student\_averages.items():

        if avg >= 90:

            grade = 'A'

        elif avg >= 80:

            grade = 'B'

        elif avg >= 70:

            grade = 'C'

        elif avg >= 60:

            grade = 'D'

        else:

            grade = 'F'

        grades[reg\_no] = (avg, grade)

    return grades

**# Run mapper and reducer**

student\_averages = mapper('student\_marks.txt')

student\_grades = reducer(student\_averages)

**# Display results**

for reg\_no, (avg, grade) in student\_grades.items():

    print(f"Reg.No: {reg\_no}, Average Marks: {avg:.2f}, Grade: {grade}")

**OUTPUT:**

Reg.No: 101, Average Marks: 86.60, Grade: B

Reg.No: 102, Average Marks: 70.00, Grade: C

Reg.No: 103, Average Marks: 58.80, Grade: F

**Lab 5: Develop a MapReduce program to implement Matrix Multiplication.**

**For Runing the environment**

!pip install mrjob

**Input Data**

with open("matrix\_input.txt", "w") as f:

    f.write("""A,0,0,1

A,0,1,2

A,1,0,3

A,1,1,4

B,0,0,5

B,1,0,6

B,0,1,7

B,1,1,8

""")

**For matrix Multiplication**

%%writefile matrix\_multiplication.py

from mrjob.job import MRJob

class MRMatrixMultiplication(MRJob):

    def configure\_args(self):

        super(MRMatrixMultiplication, self).configure\_args()

        self.add\_passthru\_arg('--m', type=int, help="Rows in A")

        self.add\_passthru\_arg('--n', type=int, help="Cols in A / Rows in B")

        self.add\_passthru\_arg('--p', type=int, help="Cols in B")

    def mapper(self, \_, line):

        matrix, i, j, value = line.strip().split(",")

        i, j, value = int(i), int(j), float(value)

        if matrix == "A":

            for k in range(self.options.p):

                yield (i, k), ('A', j, value)

        else:

            for k in range(self.options.m):

                yield (k, j), ('B', i, value)

    def reducer(self, key, values):

        A\_vals = {}

        B\_vals = {}

        for matrix, index, value in values:

            if matrix == 'A':

                A\_vals[index] = value

            else:

                B\_vals[index] = value

        total = 0

        for j in range(self.options.n):

            total += A\_vals.get(j, 0) \* B\_vals.get(j, 0)

        yield key, total

if \_\_name\_\_ == '\_\_main\_\_':

    MRMatrixMultiplication.run()

**For Printing the Output**

from matrix\_multiplication import MRMatrixMultiplication

import sys

from io import StringIO

import numpy as np

**# Simulate command-line args**

sys.argv = ['matrix\_multiplication.py', 'matrix\_input.txt', '--m', '2', '--n', '2', '--p', '2']

# Parse the arguments to get m, n, and p

m = int(sys.argv[sys.argv.index('--m') + 1])

n = int(sys.argv[sys.argv.index('--n') + 1])

p = int(sys.argv[sys.argv.index('--p') + 1])

**# Initialize matrices A and B from the input file**

A = [[0 for \_ in range(n)] for \_ in range(m)]

B = [[0 for \_ in range(p)] for \_ in range(n)]

with open("matrix\_input.txt", "r") as f:

for line in f:

matrix, i, j, value = line.strip().split(",")

i, j, value = int(i), int(j), float(value)

if matrix == "A":

A[i][j] = value

else:

B[i][j] = value

**# Run MRJob inline and capture output**

mr\_job = MRMatrixMultiplication(args=sys.argv[1:])

results = []

with mr\_job.make\_runner() as runner:

runner.run()

for line in mr\_job.parse\_output(runner.cat\_output()):

results.append(line)

**# Initialize result matrix C with zeros using the extracted m and p**

C = [[0 for \_ in range(p)] for \_ in range(m)]

for (i, j), val in results:

C[i][j] = val # Corrected the index to use both i and j

**# Pretty print all matrices**

print("🟦 Matrix A =")

for row in A:

print(row)

print("\n🟦 Matrix B =")

for row in B:

print(row)

print("\n🟩 Matrix C = A × B =")

for row in C:

print([round(val, 2) for val in row]) # rounding for neatness

**OUTPUT:**

🟦 Matrix A =

[1.0, 2.0]

[3.0, 4.0]

🟦 Matrix B =

[5.0, 7.0]

[6.0, 8.0]

🟩 Matrix C = A × B =

[17.0, 23.0]

[39.0, 53.0]

**Lab 6: Develop a MapReduce to find the maximum electrical consumption in each year given electrical consumption for each month in each year.**

**Install mrjob python**

!pip install mrjob

**Input Format (text file)**

with open("consumption\_input.txt", "w") as f:

f.write("""2019,Jan,350

2019,Feb,420

2019,Mar,390

2020,Jan,510

2020,Feb,475

2020,Mar,525

2021,Jan,600

2021,Feb,580

2021,Mar,610

""")

**The MapReduce Code**

%%writefile max\_consumption.py

from mrjob.job import MRJob

class MRMaxConsumptionPerYear(MRJob):

def mapper(self, \_, line):

# Split line into parts: year, month, value

year, month, consumption = line.strip().split(",")

yield year, int(consumption)

def reducer(self, year, consumptions):

# Return the maximum consumption for the year

yield year, max(consumptions)

if \_\_name\_\_ == '\_\_main\_\_':

MRMaxConsumptionPerYear.run()

**Run the MapReduce Job and Show Output**

from max\_consumption import MRMaxConsumptionPerYear

import sys

**# Simulate command-line input**

sys.argv = ['max\_consumption.py', 'consumption\_input.txt']

**# Run the job**

mr\_job = MRMaxConsumptionPerYear(args=sys.argv[1:])

with mr\_job.make\_runner() as runner:

runner.run()

print("🔋 Maximum Consumption Per Year:")

for line in mr\_job.parse\_output(runner.cat\_output()):

year, max\_val = line

print(f"{year} → {max\_val} kWh")

**Output**

🔋 Maximum Consumption Per Year:

2019 → 520 kWh

2021 → 610 kWh

2020 → 525 kWh

**Lab 7: Develop a MapReduce to analyze weather data set and print whether the day is shinny or cool day.**

**Installing the MAPREDUCE**

!pip install mrjob

**Input text File**

with open("weather\_input.txt", "w") as f:

    f.write("""2023-06-20,35,Clear

2023-06-21,22,Rain

2023-06-22,30,Cloudy

2023-06-23,18,Fog

2023-06-24,29,Clear

2023-06-25,20,Cloudy

""")

**The Mapreduce Program Code**

%%writefile weather\_analysis.py

from mrjob.job import MRJob

class MRWeatherAnalysis(MRJob):

    def mapper(self, \_, line):

        date, temp, condition = line.strip().split(",")

        temp = int(temp)

        if temp >= 30 or condition.lower() == "clear":

            yield date, "Sunny day"

        elif temp < 25 or condition.lower() in ["rain", "fog"]:

            yield date, "Cool day"

        else:

            yield date, "Normal day"

    def reducer(self, date, types):

        # Since there's only one type per date, just return it

        for day\_type in types:

            yield date, day\_type

            break

if \_\_name\_\_ == '\_\_main\_\_':

    MRWeatherAnalysis.run()

**To Printing of the file**

from weather\_analysis import MRWeatherAnalysis

import sys

sys.argv = ['weather\_analysis.py', 'weather\_input.txt']

mr\_job = MRWeatherAnalysis(args=sys.argv[1:])

with mr\_job.make\_runner() as runner:

    runner.run()

    print("🌦️ Weather Summary:")

    for line in mr\_job.parse\_output(runner.cat\_output()):

        date, result = line

        print(f"{date} → {result}")

**Output**

🌦️ Weather Summary:

2023-06-20 → Sunny day

2023-06-21 → Cool day

2023-06-24 → Sunny day

2023-06-25 → Cool day

2023-06-22 → Sunny day

2023-06-23 → Cool day

**Lab 8: Develop a MapReduce program to find the number of products sold in each country by considering sales data containing fields**

**Step 1: Create a sample sales data CSV file**

file\_name = "sales\_data.csv"

with open(file\_name, "w") as file:

    file.write("OrderID,ProductName,Country,Quantity\n")

    file.write("1001,Pen,USA,5\n")

    file.write("1002,Notebook,India,3\n")

    file.write("1003,Pen,India,2\n")

    file.write("1004,Pencil,USA,1\n")

    file.write("1005,Marker,UK,4\n")

    file.write("1006,Eraser,India,6\n")

    file.write("1007,Pen,Germany,7\n")

    file.write("1008,Notebook,India,2\n")

print(f"✅ Input file '{file\_name}' created with sample data.")

**Step 2: Simulate MapReduce - with Map and Reduce phases**

from collections import defaultdict

**# 🔹 Map Phase: Emit (country, quantity)**

def mapper(filename):

    map\_output = []

    with open(filename, "r") as file:

        next(file)  # Skip header

        for line in file:

            fields = line.strip().split(",")

            if len(fields) == 4:

                country = fields[2]

                quantity = int(fields[3])

                map\_output.append((country, quantity))

    return map\_output

**# 🔹 Shuffle & Sort Phase: Group by country**

def shuffle\_sort(mapped\_data):

    grouped = defaultdict(list)

    for country, quantity in mapped\_data:

        grouped[country].append(quantity)

    return grouped

**# 🔹 Reduce Phase: Sum quantities per country**

def reducer(grouped\_data):

    reduced\_output = {}

    for country, quantities in grouped\_data.items():

        reduced\_output[country] = sum(quantities)

    return reduced\_output

**# 🔁 Run the full MapReduce process**

mapped\_data = mapper(file\_name)

print("🔹 Mapper Output:")

print(mapped\_data)

grouped\_data = shuffle\_sort(mapped\_data)

print("\n🔹 Grouped Data (Shuffle & Sort):")

for country, values in grouped\_data.items():

    print(f"{country} → {values}")

reduced\_result = reducer(grouped\_data)

print("\n🧾 Final Output (Reducer Result):")

for country, total in reduced\_result.items():

    print(f"{country}: {total} units")

**Mapper Output:**

[('USA', 5), ('India', 3), ('India', 2), ('USA', 1), ('UK', 4), ('India', 6), ('Germany', 7), ('India', 2)]

**🔹 Grouped Data (Shuffle & Sort):**

USA → [5, 1]

India → [3, 2, 6, 2]

UK → [4]

Germany → [7]

**🧾 Final Output (Reducer Result):**

USA: 6 units

India: 13 units

UK: 4 units

Germany: 7 units

**Lab 9: Develop a MapReduce program to find the tags associated with each movie by analyzing movie lens data.**

**# Upload file from your local system**

from google.colab import files

uploaded = files.upload()

Step 2: Import Libraries & Load File

import pandas as pd

from collections import defaultdict

# Load the uploaded file into a DataFrame

df = pd.read\_csv('tags.csv')

# Show the first few rows to verify

df.head()

**Step 3: Simulate MapReduce to Get Tags for Each Movie**

**# 🔹 Mapper Phase: Extract (movieId, tag) pairs**

def mapper(df):

mapped = []

for \_, row in df.iterrows():

movie\_id = row['movieId']

tag = row['tag']

mapped.append((movie\_id, tag))

return mapped

**# 🔹 Shuffle & Sort: Group tags by movieId**

def shuffle\_sort(mapped\_data):

grouped = defaultdict(list)

for movie\_id, tag in mapped\_data:

grouped[movie\_id].append(tag)

return grouped

**# 🔹 Reducer Phase: Return grouped tags**

def reducer(grouped\_data):

return grouped\_data

**# Run the simulation**

mapped\_data = mapper(df)

print("🔹 Mapper Output (First 10):")

print(mapped\_data[:10])

grouped\_data = shuffle\_sort(mapped\_data)

reduced\_output = reducer(grouped\_data)

Step 4: Display Final Output (MovieID → Tags)

# 🎬 Final Result

print("\n🎬 MovieID → List of Tags (First 10 Movies):\n")

for movie\_id, tags in list(reduced\_output.items())[:10]:

print(f"{movie\_id}: {tags}")

**Sample Output**

MovieID → List of Tags (First 10 Movies):

60756: ['funny', 'Highly quotable', 'will ferrell', 'comedy', 'funny', 'will ferrell', 'funny', 'will ferrell']

89774: ['Boxing story', 'MMA', 'Tom Hardy']

106782: ['drugs', 'Leonardo DiCaprio', 'Martin Scorsese', 'Stock Market', 'Wall Street']

48516: ['way too long', 'Leonardo DiCaprio', 'suspense', 'twist ending', 'undercover cop', 'atmospheric', 'Jack Nicholson', 'Leonardo DiCaprio', 'Martin Scorsese', 'suspense']

431: ['Al Pacino', 'gangster', 'mafia']

1221: ['Al Pacino', 'Mafia', 'Mafia']

5995: ['holocaust', 'true story', 'Holocaust']

44665: ['twist ending']

52604: ['Anthony Hopkins', 'courtroom drama', 'twist ending']

88094: ['britpop', 'indie record label', 'music']

**Lab 10: XYZ.com is an online music website where users listen to various tracks, the data getscollected which is given,**

**Step 1 – Input File Creation**

user\_id,track\_id,track\_name,duration\_seconds,timestamp

U001,T001,Song A,210,2025-06-25 10:01:00

U002,T002,Song B,180,2025-06-25 10:05:00

U001,T001,Song A,210,2025-06-25 11:00:00

U003,T003,Song C,240,2025-06-25 11:15:00

U002,T001,Song A,210,2025-06-25 11:30:00

U003,T002,Song B,180,2025-06-25 12:00:00

U004,T001,Song A,210,2025-06-25 12:15:00

Save the following sample as a CSV file (music\_listening\_data.csv):

**Step 2: Upload and Analyze**

**# STEP 1: Import required libraries**

import pandas as pd

from google.colab import files

**# STEP 2: Upload the CSV file**

uploaded = files.upload()

**# STEP 3: Load data into pandas DataFrame**

df = pd.read\_csv('music\_listening\_data.csv')

print("Data Preview:")

print(df.head())

**# STEP 4: Track-wise play count**

track\_play\_count = df['track\_name'].value\_counts()

print("\n🔹 Number of Plays per Track:")

print(track\_play\_count)

**# STEP 5: Unique listeners per track**

unique\_users\_per\_track = df.groupby('track\_name')['user\_id'].nunique()

print("\n🔹 Unique Users per Track:")

print(unique\_users\_per\_track)

**# STEP 6: Total listening duration per track**

total\_duration = df.groupby('track\_name')['duration\_seconds'].sum()

print("\n🔹 Total Listening Time (in seconds) per Track:")

print(total\_duration)

**# STEP 7: Total duration listened per user**

user\_duration = df.groupby('user\_id')['duration\_seconds'].sum()

print("\n🔹 Total Listening Time per User:")

print(user\_duration)

**Lab 11:Develop a MapReduce program to find the frequency of books published eachyear and find in which year maximum number of books were published usingthe given data.**

**Books.csv**

|  |  |  |  |
| --- | --- | --- | --- |
| book\_id | book\_title | author | publication\_year |
| B001 | Data Science 101 | John Smith | 2019 |
| B002 | Big Data Basics | Alice Ray | 2020 |
| B003 | Machine Learning | Amit Shah | 2018 |
| B004 | AI for Everyone | Karen Jones | 2020 |
| B005 | Cloud Computing | Ravi Kumar | 2019 |
| B006 | Deep Learning | John Smith | 2020 |
| B007 | Hadoop in Action | Alice Ray | 2019 |

**Uploading the Books.csv file**

from google.colab import files

uploaded = files.upload()

**Importing the Pandas**

import pandas as pd

from collections import defaultdict

**# Step 1: Load the dataset**

file\_path = 'books.csv' # upload this CSV file first in Colab

df = pd.read\_csv(file\_path)

**# Step 2: Simulate Mapper - Emit (year, 1)**

mapped = [(year, 1) for year in df['publication\_year']]

**# Step 3: Shuffle & Sort (Group by year)**

shuffled = defaultdict(list)

for year, count in mapped:

shuffled[year].append(count)

**# Step 4: Reducer - Count books per year**

reduced = {year: sum(counts) for year, counts in shuffled.items()}

**# Print the results**

print("📚 Number of books published each year:")

for year in sorted(reduced):

print(f"{year}: {reduced[year]} books")

**# Step 5: Find year with maximum publications**

max\_year = max(reduced, key=reduced.get)

print(f"\n📈 Year with maximum number of books published: {max\_year} ({reduced[max\_year]} books)")

**Lab 12: Develop a MapReduce program to analyze Titanic ship data and to find the average age of the people (both male and female) who died in the tragedy. How many persons are survived in each class.**

**Input File**

PassengerId,Survived,Pclass,Name,Sex,Age

1,0,3,Braund, Mr. Owen Harris,male,22

2,1,1,Cumings, Mrs. John Bradley,female,38

3,1,3,Heikkinen, Miss. Laina,female,26

4,1,1,Futrelle, Mrs. Jacques Heath,female,35

5,0,3,Allen, Mr. William Henry,male,35

6,0,3,Moran, Mr. James,male,

7,0,1,McCarthy, Mr. Timothy J,male,54

8,1,3,Palsson, Miss. Torborg Danira,female,8

9,1,3,Johnson, Master. Harold Godfrey,male,19

10,0,2,Nasser, Mrs. Nicholas (Adele Achem),female,14

titanic.csv

**Import Panda**

import pandas as pd

from collections import defaultdict

**# Step 1: Load the dataset**

df = pd.read\_csv("titanic.csv")

**# Step 2: Clean data**

df = df.dropna(subset=['Age']) # Remove rows where age is missing

**# Step 3: Mapper - Emit (sex, age) for passengers who died**

death\_age = df[df['Survived'] == 0][['Sex', 'Age']]

mapped\_age = defaultdict(list)

for \_, row in death\_age.iterrows():

mapped\_age[row['Sex']].append(row['Age'])

**# Step 4: Reducer - Calculate average age by sex**

print("☠️ Average age of people who died (by gender):")

for sex, ages in mapped\_age.items():

avg\_age = sum(ages) / len(ages)

print(f"{sex.capitalize()}: {avg\_age:.2f} years")

**# Step 5: Mapper for survival count per class**

survived\_class = df[df['Survived'] == 1]['Pclass'].value\_counts().sort\_index()

**# Step 6: Output number of survivors per class**

print("\n🚢 Number of persons survived in each class:")

for pclass, count in survived\_class.items():

print(f"Class {pclass}: {count} survived")

**Output:**

Average age of people who died (by gender):

Male: 37.00 years

Female: 14.00 years

🚢 Number of persons survived in each class:

Class 1: 2 survived

Class 3: 3 survived

**Lab 13: Develop a MapReduce program to analyze Uber data set to find the days on which each basement has more trips using the given dataset.**

Step 1: Sample Dataset uber.csv

**Input File**

Date,Base,Trips

2014-09-01,B02512,190

2014-09-01,B02598,113

2014-09-01,B02617,215

2014-09-02,B02512,200

2014-09-02,B02598,120

2014-09-02,B02617,220

2014-09-03,B02512,180

2014-09-03,B02598,115

2014-09-03,B02617,230

from google.colab import files

uploaded = files.upload()

**Step 2: Python MapReduce Program**

import pandas as pd

from collections import defaultdict

**# Step 1: Load dataset**

df = pd.read\_csv('uber.csv')

**# Step 2: Mapper - Emit (Base, (Date, Trips))**

mapped = defaultdict(list)

for \_, row in df.iterrows():

mapped[row['Base']].append((row['Date'], row['Trips']))

**# Step 3: Reducer - For each base, find the date(s) with maximum trips**

print("🚗 Days with highest trips for each Uber base:\n")

for base, records in mapped.items():

max\_trips = max(trips for \_, trips in records)

max\_days = [date for date, trips in records if trips == max\_trips]

print(f"Base {base}:")

for day in max\_days:

print(f" - {day} ({max\_trips} trips)")

**Lab 14:Develop a program to calculate the maximum recorded temperature by yearwise for the weather dataset in Pig Latin**

**Input File**

USC0001 19810101 35

USC0002 19810102 38

USC0001 19820101 40

USC0002 19820103 44

USC0001 19830101 29

USC0002 19830102 32

**-- Step 1: Load the data (assuming space-delimited)**

weather\_data = LOAD 'weather\_data.txt'

USING PigStorage(' ')

AS (station:chararray, date:chararray, temperature:int);

**-- Step 2: Extract year from date**

weather\_with\_year = FOREACH weather\_data GENERATE

SUBSTRING(date, 0, 4) AS year, temperature;

**-- Step 3: Group by year**

grouped\_by\_year = GROUP weather\_with\_year BY year;

**-- Step 4: Find maximum temperature per year**

max\_temp\_by\_year = FOREACH grouped\_by\_year GENERATE

group AS year,

MAX(weather\_with\_year.temperature) AS max\_temp;

**-- Step 5: Store/Display output**

DUMP max\_temp\_by\_year;

**-- Or use STORE to write to file:**

**-- STORE max\_temp\_by\_year INTO 'output\_directory' USING PigStorage(',');**

**Lab 15: Write queries to sort and aggregate the data in a table using HiveQL.**

**INPUT File**

| **sale\_id** | **product** | **category** | **quantity** | **price** | **sale\_date** |
| --- | --- | --- | --- | --- | --- |
| 1 | Pen | Stationery | 10 | 5.0 | 2025-01-01 |
| 2 | Notebook | Stationery | 15 | 20.0 | 2025-01-02 |
| 3 | Pencil | Stationery | 50 | 2.0 | 2025-01-01 |
| 4 | Chair | Furniture | 5 | 150.0 | 2025-01-03 |
| 5 | Table | Furniture | 2 | 300.0 | 2025-01-03 |

**-- Create the table**

CREATE TABLE IF NOT EXISTS sales\_data (

sale\_id INT,

product STRING,

category STRING,

quantity INT,

price FLOAT,

sale\_date STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

**-- Load data into the table (assume a file 'sales\_data.csv' is in HDFS)**

LOAD DATA INPATH '/user/hive/warehouse/sales\_data.csv' INTO TABLE sales\_data;

**-- 1️⃣ Sort all records by price (ascending)**

SELECT \* FROM sales\_data

ORDER BY price ASC;

**-- 2️⃣ Sort by quantity (descending)**

SELECT \* FROM sales\_data

ORDER BY quantity DESC;

**-- 3️⃣ Aggregate: Total quantity sold per category**

SELECT category, SUM(quantity) AS total\_quantity

FROM sales\_data

GROUP BY category;

**-- 4️⃣ Aggregate: Average price per category**

SELECT category, AVG(price) AS average\_price

FROM sales\_data

GROUP BY category;

**-- 5️⃣ Aggregate: Total revenue (price \* quantity) per product**

SELECT product, SUM(price \* quantity) AS total\_revenue

FROM sales\_data

GROUP BY product;

**-- 6️⃣ Count number of sales per day**

SELECT sale\_date, COUNT(\*) AS sale\_count

FROM sales\_data

GROUP BY sale\_date

ORDER BY sale\_date;