Assignment 2 – Hive and Spark

Author – Aditya Saxena

Contents

[Executive Summary 3](#_Toc193702699)

[The Dataset 4](#_Toc193702700)

[Part 1 – Hadoop and Hive 6](#_Toc193702701)

[Environment Setup using Oracle VM VirtualBox for Hadoop and Hive 6](#_Toc193702702)

[Part 2 – Apache Spark 11](#_Toc193702703)

[Environment Setup using Oracle VM VirtualBox for Hadoop and Hive 11](#_Toc193702704)

# Executive Summary

**This assignment helps the student gain practical, hands-on skills in Big Data tools and technologies**, focusing on data ingestion, processing, analysis, and querying using Hadoop, Hive, and Apache Spark.

**💾 Big Data Environment Setup & Management**

* Configured and managed virtual machines (VMs) using Oracle VirtualBox.
* Established secure access via SSH using PuTTY.
* Transferred data files between host systems and VMs using SCP tools like pscp.

**📂 Working with Hadoop Distributed File System (HDFS)**

* Navigated and manipulated files within HDFS using commands like put, get, ls, cp, and rm.
* Managed datasets in a distributed environment, preparing them for downstream processing.

**🐘 Data Warehousing with Apache Hive**

* Created, queried, and managed Hive databases and tables.
* Executed complex HiveQL queries involving filtering, aggregation, joins, and nested queries.
* Handled advanced data types such as arrays, maps, and structs in Hive.
* Applied user-defined aggregate functions (UDAFs) for natural language processing tasks (e.g., tokenizing and extracting n-grams).
* Transferred structured data between MySQL and Hive using **Apache Sqoop**.

**⚙️ Big Data Processing with Apache Spark**

* Developed applications using both the **Resilient Distributed Dataset (RDD)** and **DataFrame** APIs.
* Applied Spark transformations like map, flatMap, filter, reduceByKey, and actions like collect, take, and show.
* Wrote and executed PySpark jobs for word count and data summarization tasks.
* Leveraged **Spark SQL** to query large datasets using SQL-like syntax.
* Learned how to build and run **Spark batch applications** with spark-submit.

**📊 Analytical Thinking & Problem Solving**

* Gained experience designing end-to-end data workflows: ingest → process → analyze → visualize.
* Practiced writing efficient and scalable queries to extract meaningful insights from large datasets.
* Developed confidence in working with semi-structured and unstructured data using real-world datasets (e.g., tweets, logs).

## The Dataset

**MovieLens Dataset – File Descriptions**

This dataset contains three data files used for analyzing user ratings and movie information. The details of each file are as follows:

**1. u.data**

* **Description:**  
  A tab-separated (\t) list of **100,000 ratings** provided by **943 users** on **1,682 items (movies)**.  
  Users and items are numbered consecutively.
* **Columns:**
  + User ID
  + Item ID
  + Rating
  + Timestamp

**2. u.user**

* **Description:**  
  A list of **943 users** with demographic information.  
  Values in each row are separated by a **pipe symbol (|)**.
* **Columns:**
  + User ID
  + Age
  + Gender
  + Occupation
  + Zip Code

**3. u.item**

* **Description:**  
  Contains information about the items (movies).  
  Each row is separated by a **pipe symbol (|)**.
* **Columns:**
  + Movie ID
  + Movie Title
  + Release Date
  + Video Release Date
  + IMDb URL
  + Unknown
  + Action
  + Adventure
  + Animation
  + Children's
  + Comedy
  + Crime
  + Documentary
  + Drama
  + Fantasy
  + Film-Noir
  + Horror
  + Musical
  + Mystery
  + Romance
  + Sci-Fi
  + Thriller
  + War
  + Western

The last 19 fields represent movie genres.  
A value of **"1"** indicates the movie belongs to that genre, while **"0"** indicates it does not.  
Movies may belong to multiple genres simultaneously.

## Part 1 – Hadoop and Hive

### Environment Setup using Oracle VM VirtualBox for Hadoop and Hive

**🔧 Hortonworks Sandbox VM Setup for Hive Work**

**✅ Requirements**

* RAM: Minimum **16 GB**
* Software:
  + [Oracle VM VirtualBox](https://www.virtualbox.org/)
  + Hortonworks Sandbox with HDP **2.2.4.2** (.ova file) – available on D2L under *Lab Resources & Datasets*
  + [PuTTY](https://www.putty.org/) (for SSH access)
  + [PSCP](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) (for file transfer on Windows)

**⚙️ Step-by-Step Setup**

**1. Import the Hortonworks VM**

* Open **Oracle VM VirtualBox Manager**
* Go to File → Import Appliance
* Select the downloaded .ova file (e.g., Hortonworks\_Sandbox\_HDP\_2.2.4.2.ova)
* Import the VM and allocate **at least 8–10 GB RAM** (16 GB recommended)

**2. Start the VM**

* Select the Hortonworks VM and click **Start**
* Wait for all Hadoop system components to fully load (may take a few minutes)
* You'll see a **login prompt** on the screen

**3. Log into the VM**

* At the prompt, enter:
  + **Username:** root
  + **Password:** hadoop

**4. Set Up PuTTY for SSH Access (on Windows)**

* Open **PuTTY**
* In the **Host Name** field, enter: 127.0.0.1
* **Port:** 2222
* Save the session with a name like HDP Sandbox
* Click **Open**, then log in:
  + **Username:** root
  + **Password:** hadoop

✅ *Now you're connected to the HDP access node via PuTTY.*

**📁 5. Transfer Hive Dataset to the Access Node**

Use **PSCP** to copy the required file (e.g., full\_text.txt) from Windows to the VM:

pscp -P 2222 -pw hadoop full\_text.txt root@127.0.0.1:/home/cind719/

📝 *Replace full\_text.txt with the actual file name.*

**📦 6. Move the Dataset into HDFS**

In the **PuTTY terminal**, run the following commands:

# Create a directory in HDFS

hadoop fs -mkdir /user/lab

# Move the file into HDFS

hadoop fs -put /home/cind719/full\_text.txt /user/lab

# Confirm it was uploaded

hadoop fs -ls /user/lab

**🐝 7. Launch Hive CLI**

Once the file is in HDFS:

hive

This will take you to the **Hive CLI** (hive> prompt), where you can now:

* Create databases and tables
* Load data from HDFS
* Write and run HiveQL queries

Example:

CREATE DATABASE demo;

USE demo;

CREATE TABLE tweets (

id STRING,

ts STRING,

lat STRING,

lon STRING,

tweet STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t';

LOAD DATA INPATH '/user/lab/full\_text.txt' INTO TABLE tweets;

**🧹 Optional: Clean Up**

To delete data:

hadoop fs -rm /user/lab/full\_text.txt

DROP DATABASE demo CASCADE;

## Part 2 – Apache Spark

### Environment Setup using Oracle VM VirtualBox for Hadoop and Hive

**⚙️ Setting Up Apache Spark 3.0 CU\_VM (Second VM) for Spark Labs (Lab 5+)**

This setup is used for PySpark-based labs (Lab 5 and Lab 6) where you work with Spark RDDs, DataFrames, and Spark SQL.

**✅ Requirements**

* **RAM:** 8 GB minimum (more is better)
* **Software:**
  + [Oracle VM VirtualBox](https://www.virtualbox.org/)
  + Spark 3.0 CU\_VM .ova file – available on D2L under *Lab Resources & Datasets → Apache Spark 3.0 CU\_VM.ova*
  + [PuTTY](https://www.putty.org/) for SSH access
  + [PSCP](https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html) for file transfer (Windows only)

**🪟 1. Import and Start the VM**

1. Open **Oracle VM VirtualBox**
2. Go to File → Import Appliance
3. Select the Apache Spark 3.0 CU\_VM.ova file
4. Allocate **at least 8 GB RAM** to the VM (more if available)
5. Start the VM

Let it boot fully. You’ll see a terminal login prompt when ready.

**🔐 2. Set Up PuTTY for SSH Access**

1. Open **PuTTY**
2. In the **Host Name** field: 127.0.0.1
3. In the **Port** field: 2020
4. Save the session as Spark3\_VM or similar
5. Click **Open** and log in with:
   * **Username:** cu
   * **Password:** spark

✅ You're now in the Spark 3.0 CU\_VM via PuTTY.

**📁 3. Transfer Dataset to the VM**

If you need to transfer a file (e.g., nasa\_data.csv):

pscp -P 2020 -pw spark nasa\_data.csv cu@127.0.0.1:/home/cu/data/

📝 Make sure the /home/cu/data/ directory exists. Create it if needed:

mkdir -p /home/cu/data/

**🔥 4. Launch PySpark Shell**

Once logged in via PuTTY:

pyspark

This launches the **interactive Spark shell** with the preconfigured environment.

You can now use Spark RDDs, DataFrames, and Spark SQL.

Example:

nums = sc.parallelize([1, 2, 3, 4, 5])

nums.map(lambda x: x \* x).collect()

**💼 5. Run PySpark Scripts as Batch Jobs**

To run Spark scripts like word\_counter.py or nasa\_top\_hosts.py:

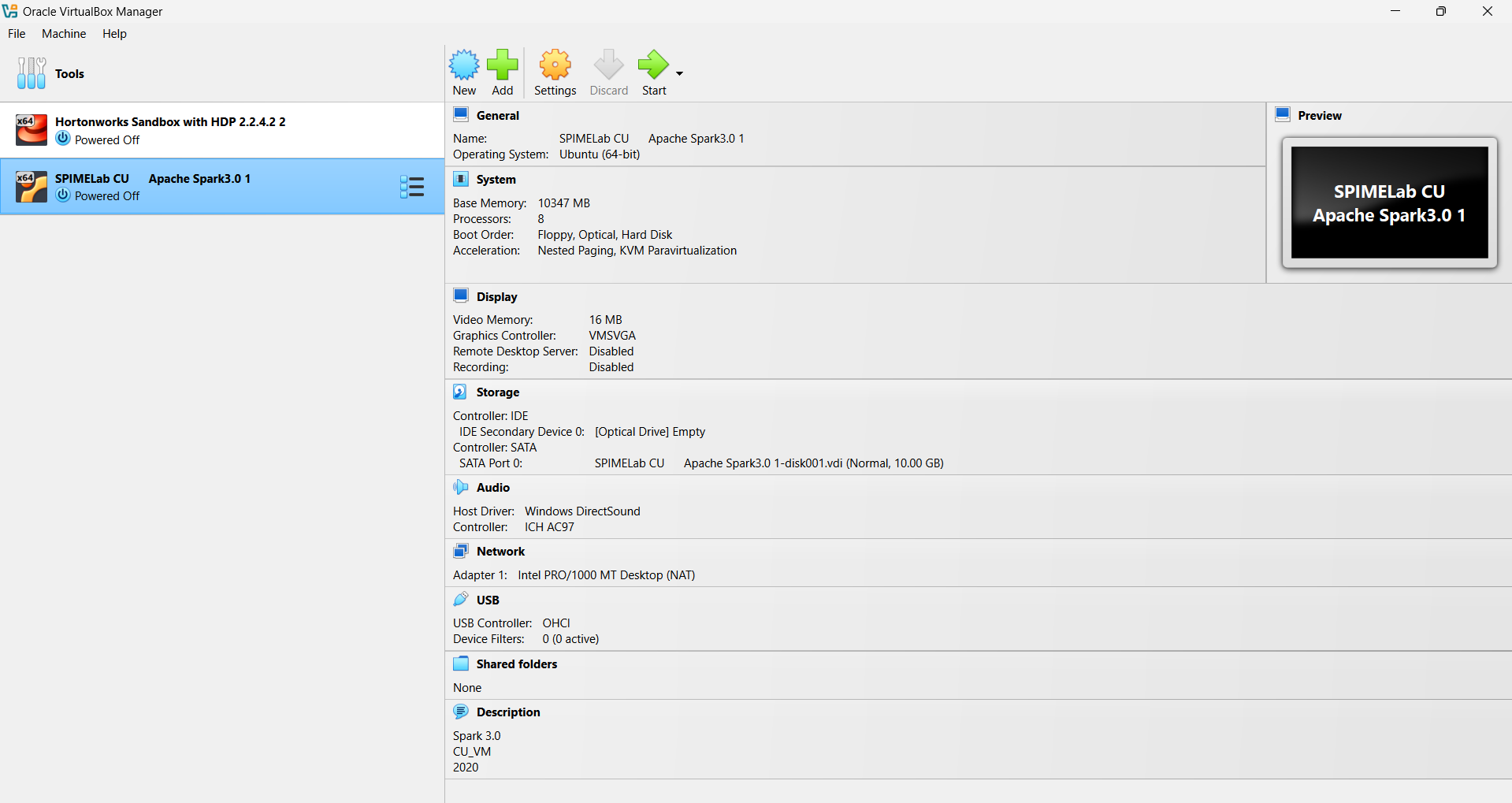
1. Create the Python script using nano or transfer it via pscp
2. Run it with:

spark-submit word\_counter.py

✅ Output will be printed to the terminal.

**🧹 Optional: Shut Down & Cleanup**

* Exit PySpark: quit()
* Shut down VM: use shutdown now inside the terminal or close it from VirtualBox



## Implementation – Hive and Spark Problems

## 💻 Setup Instructions

### 🔹 STEP 1: Transferring files from local machine to VM using `pscp`

Use the following command from your \*\*Windows Command Prompt\*\* (after placing the dataset in a known directory):

```bash

pscp -P 2020 -pw spark u.data cu@127.0.0.1:/home/cu/

pscp -P 2020 -pw spark u.user cu@127.0.0.1:/home/cu/

pscp -P 2020 -pw spark u.item cu@127.0.0.1:/home/cu/

This will transfer the dataset files to the /home/cu/ directory in the Spark 3.0 CU\_VM.

A computer screen with white text

AI-generated content may be incorrect.

**🔹 STEP 2: Confirming transfer using PuTTY**

* Open **PuTTY** and log in using:
  + Host: 127.0.0.1
  + Port: 2020
  + Username: cu
  + Password: spark
* After login, run:

ls /home/cu

You should see u.data, u.user, and u.item listed.

A screenshot of a computer screen

AI-generated content may be incorrect.

**🐝 Hive Section**

**🔸 Q2.1: Store the data in a Hive database movielens as tables**

**1. Launch Hive CLI**

hive

**2. Create database**

CREATE DATABASE movielens;

USE movielens;

A black screen with white text

AI-generated content may be incorrect.

A screen shot of a movie

AI-generated content may be incorrect.

**3. Create Hive tables**

CREATE TABLE userratings (

user\_id INT,

item\_id INT,

rating INT,

timestamp BIGINT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '\t';

A screenshot of a computer program

AI-generated content may be incorrect.

CREATE TABLE users (

user\_id INT,

age INT,

gender STRING,

occupation STRING,

zip\_code STRING

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '|';

A screen shot of a computer program

AI-generated content may be incorrect.

CREATE TABLE items (

movie\_id INT,

title STRING,

release\_date STRING,

video\_release\_date STRING,

imdb\_url STRING,

unknown INT,

action INT,

adventure INT,

animation INT,

children INT,

comedy INT,

crime INT,

documentary INT,

drama INT,

fantasy INT,

film\_noir INT,

horror INT,

musical INT,

mystery INT,

romance INT,

sci\_fi INT,

thriller INT,

war INT,

western INT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY '|';



**4. Load data into tables**

LOAD DATA LOCAL INPATH '/home/cu/u.data' INTO TABLE userratings;

A screen shot of a computer

AI-generated content may be incorrect.

LOAD DATA LOCAL INPATH '/home/cu/u.user' INTO TABLE users;

A screen shot of a computer

AI-generated content may be incorrect.

LOAD DATA LOCAL INPATH '/home/cu/u.item' INTO TABLE items;

A black screen with white text

AI-generated content may be incorrect.

**🔸 Q2.2: Confirm number of records in all tables**

SELECT COUNT(\*) FROM userratings; -- Should return 100000

A screen shot of a computer

AI-generated content may be incorrect.

SELECT COUNT(\*) FROM users; -- Should return 943

A screen shot of a computer

AI-generated content may be incorrect.

Counting number of rows in Items

A screen shot of a computer

AI-generated content may be incorrect.

**🔸 Q2.3: Top 20 movies with most ratings from female programmers**

SELECT u.item\_id, COUNT(\*) AS rating\_count

FROM userratings u

JOIN users us ON u.user\_id = us.user\_id

WHERE us.gender = 'F' AND us.occupation = 'programmer'

GROUP BY u.item\_id

ORDER BY rating\_count DESC

LIMIT 20;

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**🔸 Q2.4: Top 10 highest-rated Sci-Fi movies with most ratings**

SELECT

i.title,

ROUND(AVG(r.rating), 2) AS avg\_rating,

COUNT(\*) AS num\_ratings

FROM userratings r

JOIN items i ON r.item\_id = i.movie\_id

WHERE i.sci\_fi = 1

GROUP BY i.title

HAVING COUNT(\*) > 1

ORDER BY avg\_rating DESC, num\_ratings DESC

LIMIT 10;

A screenshot of a computer

AI-generated content may be incorrect.

**⚙️ Spark Section**

**🔸 STEP: Start PySpark**

A screenshot of a computer

AI-generated content may be incorrect.

Starting pyspark

A computer screen with text on it

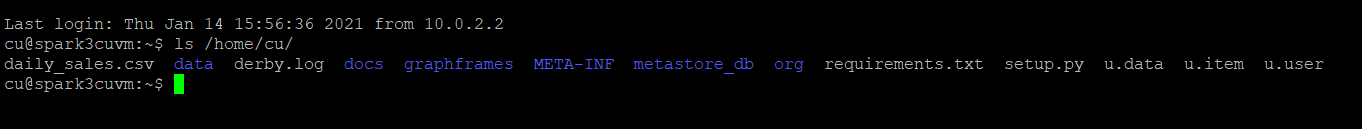
AI-generated content may be incorrect.

Transferring files from local environment to the virtual machine

A computer screen shot of a computer program

AI-generated content may be incorrect.

Confirming File Transfer to the Virtual Machine



**🔸 Q2.1: Load u.data and u.item into DataFrames**

from pyspark.sql import SparkSession

from pyspark.sql.types import StructType, StructField, IntegerType, LongType, StringType

spark = SparkSession.builder.appName("Assignment2").getOrCreate()

schema\_udata = StructType([

StructField("user\_id", IntegerType(), True),

StructField("item\_id", IntegerType(), True),

StructField("rating", IntegerType(), True),

StructField("timestamp", LongType(), True)

])

df\_udata = spark.read.csv("file:///home/cu/u.data", sep="\t", schema=schema\_udata)

df\_udata.show(5)

A screenshot of a computer

AI-generated content may be incorrect.

genre\_columns = [

"unknown", "action", "adventure", "animation", "children", "comedy",

"crime", "documentary", "drama", "fantasy", "film\_noir", "horror", "musical",

"mystery", "romance", "sci\_fi", "thriller", "war", "western"

]

schema\_uitem = StructType([

StructField("movie\_id", IntegerType(), True),

StructField("title", StringType(), True),

StructField("release\_date", StringType(), True),

StructField("video\_release\_date", StringType(), True),

StructField("imdb\_url", StringType(), True),

] + [StructField(genre, IntegerType(), True) for genre in genre\_columns])

df\_uitem = spark.read.csv("file:///home/cu/u.item", sep="|", schema=schema\_uitem)

df\_uitem.show(5)

A screenshot of a computer

AI-generated content may be incorrect.

**🔸 Q2.2: Most and least frequent genres**

from pyspark.sql.functions import sum as \_sum

# Calculate total count per genre

genre\_counts = df\_uitem.select([\_sum(genre).alias(genre) for genre in genre\_columns])

genre\_counts.show()

A black background with colorful lights

AI-generated content may be incorrect.

# Convert row to dictionary

counts = genre\_counts.first().asDict()

most\_frequent = max(counts, key=counts.get)

least\_frequent = min(counts, key=counts.get)

print(f"Most Frequent Genre: {most\_frequent} → {counts[most\_frequent]}")

print(f"Least Frequent Genre: {least\_frequent} → {counts[least\_frequent]}")

A computer screen shot of a program

AI-generated content may be incorrect.

**🔸 Q2.3: Average rating per genre**

from pyspark.sql.functions import avg, col, round

df\_joined = df\_udata.join(df\_uitem, df\_udata.item\_id == df\_uitem.movie\_id)

for genre in genre\_columns:

genre\_df = df\_joined.filter(col(genre) == 1)

avg\_rating = genre\_df.agg(round(avg("rating"), 2)).first()[0]

print(f"{genre.capitalize()}: {avg\_rating}")

A computer screen shot of a black screen

AI-generated content may be incorrect.