HADOOP

Question 1: Data Ingestion:

- Create a directory in HDFS and transfer the banking dataset from the local system to the HDFS directory.

Answer:

Steps: 1. Access the Hadoop Environment

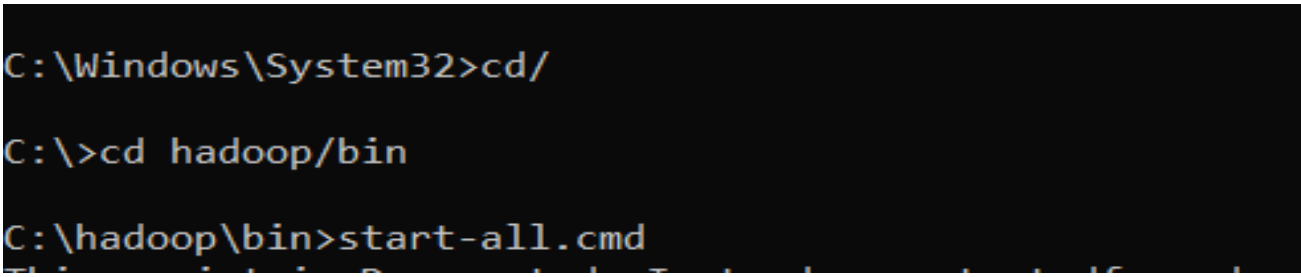
- 1. Set up the Hadoop environment:

We did this using/installing **local hadoop** as well as using **Hadoop in a Docker environment**.

1. **Setting up hadoop in local environment:**

   1. Open a terminal on the system where Hadoop is installed. Or you can directly go to hadoop bin using the command line.

   2. If you're running Hadoop in a Windows environment, you can use **start-all.cmd** to launch the necessary Hadoop services(NameNode, DataNode, ResourceManager, and NodeManager). Or you can use **start-dfs.cmd** and **start-yarn.cmd** to run the yarn demons.



2. **Create a Directory in HDFS:**

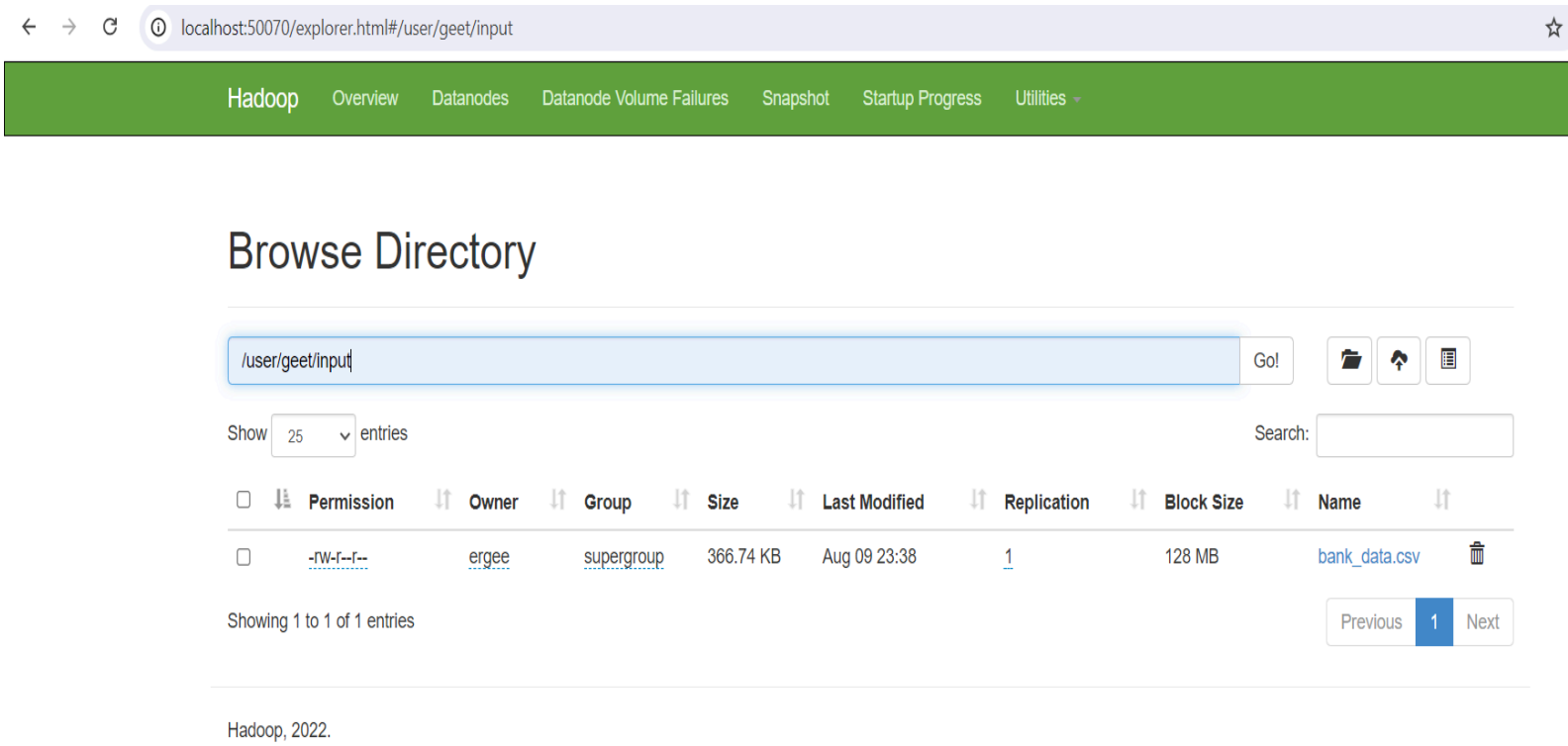
   Use the **hadoop fs -mkdir** command to create a directory in HDFS.

3. **Transfer the Banking Dataset from Local to HDFS:**

   Use the **hadoop fs -copyFromLocal** command to transfer the **bank.csv** dataset from your local file system to the newly created HDFS directory.

```
C:\hadoop\bin>hadoop fs -copyFromLocal "C:\Users\ergee\Downloads\bank_data.csv" /user/geet/input/

C:\hadoop\bin>
```

4. **Check if data is loaded to the new directory:** We can visualise the data in our localhost by accessing localhost**:50070** in our browser.



2. Setting up Hadoop Using Hadoop docker container:

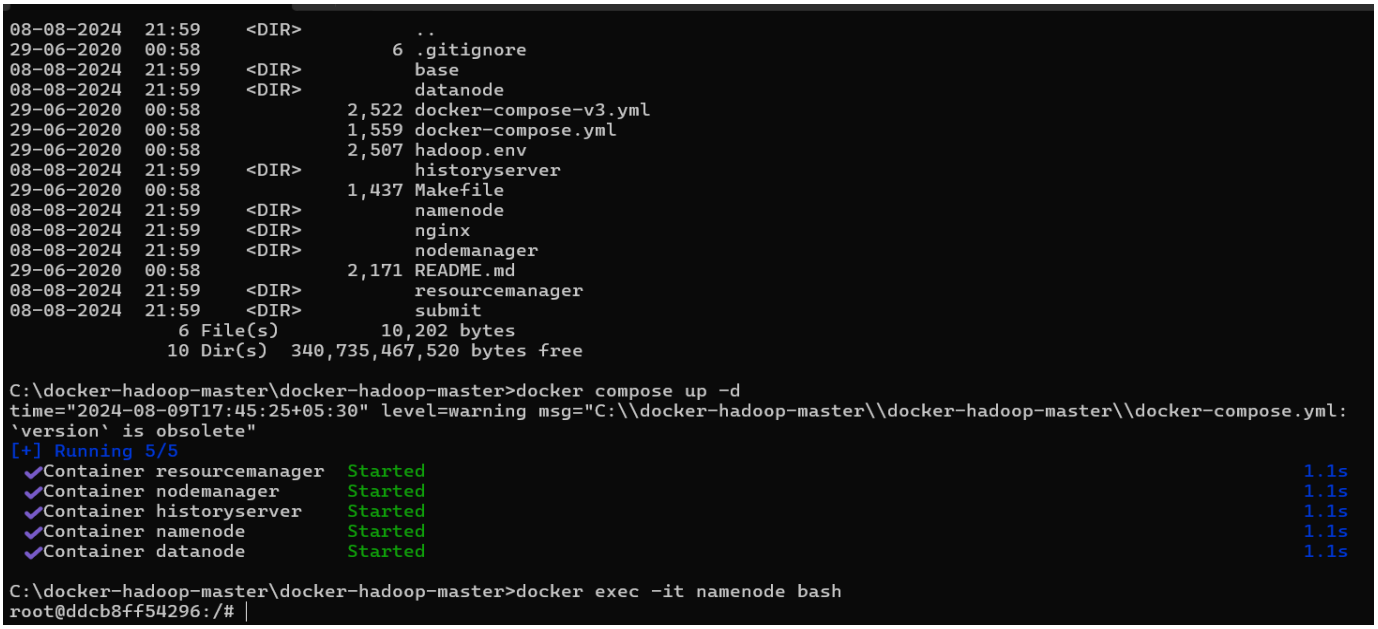
Steps to create a directory in HDFS and transfer the banking dataset from the local system to the HDFS directory using a Hadoop Docker container:

1.**Setting up Hadoop using Docker**:

1. We used the open source Framework available on git - hub to set up Hadoop.

2.Navigate to the directory containing your docker file and run the following command to start the Hadoop cluster in detached mode (background):

docker-compose up -d



Access the Container Shell:

- If you need to run HDFS commands, you can enter the running NameNode container using the following command:

  docker exec -it namenode bash

2. Create an input folder: Inside the namenode container use **hadoop fs -mkdir** command to create a new **directory.**

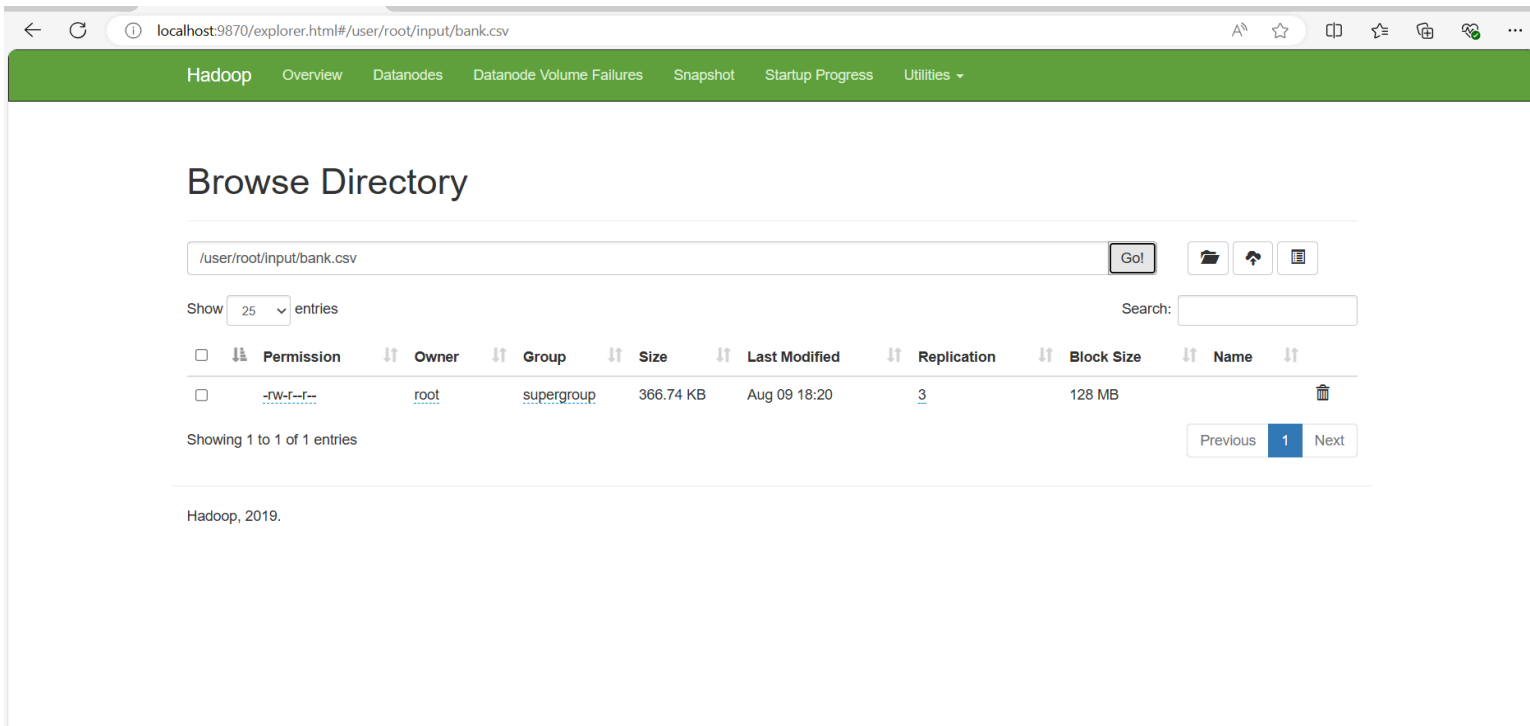
```
C:\docker-hadoop-master\docker-hadoop-master>docker exec -it namenode bash
root@ddcb8ff54296:/# hdfs dfs -mkdir /user/root/input
```

3. **Transfer the banking dataset from the local system to the HDFS directory:**

1. First move the bank.csv file from local to docker temp using **docker cp bank.csv namenode:/tmp** command.
2. Copy files from temp to the newly created input directory using **hdfs dfs -put** command.

```
C:\docker-hadoop-master\docker-hadoop-master>docker cp bank.csv namenode:/tmp
Successfully copied 377kB to namenode:/tmp

C:\docker-hadoop-master\docker-hadoop-master>docker exec -it namenode bash
root@ddcb8ff54296:/# hdfs dfs -mkdir /user/root/input
root@ddcb8ff54296:/# cd /tmp
root@ddcb8ff54296:/tmp# hdfs dfs -put bank.csv /user/root/input
2024-08-09 12:50:45,452 INFO sasl.SaslDataTransferClient: SASL encryption trust check: localHostTrusted = false, remoteHostTrusted = false
```

4. Check if data is loaded to the new directory: We can visualise the data in our localhost by accessing to **localhost:9870** in our browser.



Question 2: Data Transformation with MapReduce:

Question 2.1 **Write a MapReduce program in Python that calculates the average account balance for each job type.**

Answer:

Step: 1. Write the MapReduce Python script and save as Mapper.py and Reducer.py.

Note: These two code files (Mapper.py and Reducer.py) are submitted with this document in separate files.

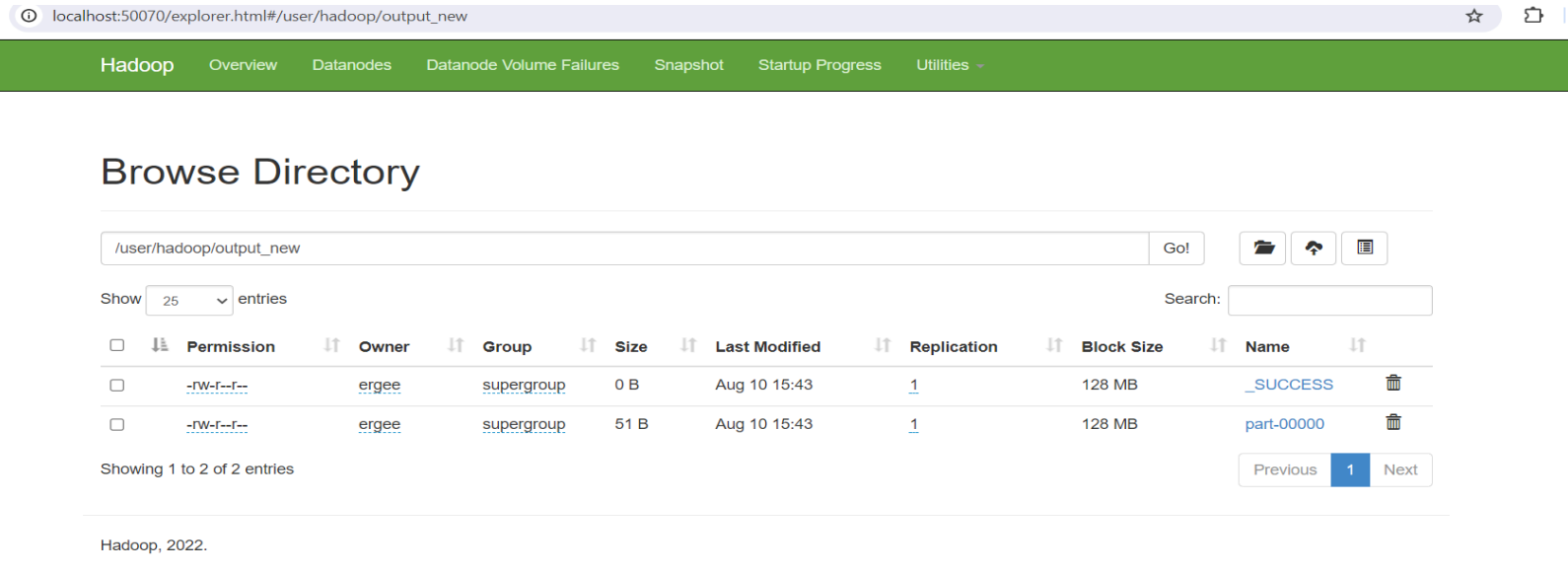
Step: 2. Create a new directory and upload the banking dataset bank_data.csv from local to HDFS.

Step: 3. Run Your MapReduce Job by specifying the input and output paths.

```
C:\hadoop\bin>hadoop fs -mkdir -p /user/geet/input/

C:\hadoop\bin>hadoop fs -copyFromLocal "C:\Users\ergee\Downloads\bank_data.csv" /user/geet/input/

C:\hadoop\bin>hadoop fs -mkdir -p /user/hadoop/input

C:\hadoop\bin>hadoop fs -copyFromLocal "C:\Users\ergee\Desktop\hadoop\mapper.py" /user/hadoop/input

C:\hadoop\bin>hadoop fs -copyFromLocal "C:\Users\ergee\Desktop\hadoop\reducer.py" /user/hadoop/input

C:\hadoop\bin>hadoop jar "C:\hadoop\share\hadoop\tools\lib\hadoop-streaming-2.10.2.jar" \    -files /user/hadoop/input/mapper.py,/user/hadoop/input/reducer.py \    -mapper "python mapper.py" \    -reducer "pytho
n reducer.py" \    -input /user/geet/input/bank_data.csv \    -output /user/hadoop/output_new
```

Step: 4. Retrieve the results from the output directory:

Use hadoop fs -cat /user/hadoop/output_new/part-00000 command to visualise and retrieve the output from the output directory.



Output:

```
C:\hadoop\bin>
C:\hadoop\bin>hadoop fs -cat /user/hadoop/output_new/part-00000
admin.  1226.73640167364
blue-collar     1085.161733615222
entrepreneur    1645.125
housemaid       2083.8035714285716
management      1766.9287925696594
retired 2319.191304347826
self-employed   1392.4098360655737
services        1103.9568345323742
student 1543.8214285714287
technician      1330.99609375
unemployed      1089.421875
unknown 1501.7105263157894
```

Question 2.2.Write another MapReduce program that counts the number of individuals with and without a housing loan in each education category.

Answer:

Step: 1. Write the MapReduce Python script and save as Mapper.py and Reducer.py.

Note: These two code files are submitted with the document in separate files.

Step: 2. Run Your MapReduce Job by specifying the input and output paths.

Note: It is generally a good practice to delete the output directory before running a new MapReduce job in Hadoop to avoid overwriting issues.

*To delete the output directory use the command **- hadoop fs -rm -r**

```
C:\hadoop\bin>hadoop fs -rm -r /user/hadoop/output_new
Deleted /user/hadoop/output_new
```

```
C:\hadoop\bin>hadoop fs -rm -r /user/hadoop/output_new
Deleted /user/hadoop/output_new

C:\hadoop\bin>hadoop jar C:\hadoop\share\hadoop\tools\lib\hadoop-streaming-2.10.2.jar -files "file:///C:/hadoop/mapper.py,file:///C:/hadoop/reducer.py" -mapper "python mapper.py" -reducer "python reducer.py" -i
put /user/geet/input/bank_data.csv\ -output /user/hadoop/output_new
```

Output:

```
C:\hadoop\bin>
C:\hadoop\bin>hadoop fs -cat /user/hadoop/output_new/part-00000
primary 93      584
secondary       416     1889
tertiary        174     1175
unknown 7       179

C:\hadoop\bin>
```

Question 2.3 - Perform a MapReduce job to determine the number of clients contacted in each month and their subscription status to term deposits ('y' column).

Answer:

Step 1 Delete the output directory before running a new MapReduce job.

Step: 2. Write the MapReduce Python script and save as Mapper.py and Reducer.py.

Note: These two code files are submitted with the document in separate files.

Step: 3. Run Your MapReduce Job by specifying the input and output paths.

```
C:\hadoop\bin>hadoop fs -rm -r /user/hadoop/output_new
Deleted /user/hadoop/output_new

C:\hadoop\bin>hadoop jar C:\hadoop\share\hadoop\tools\lib\hadoop-streaming-2.10.2.jar -files "file:///C:/hadoop/mapper.py,file:///C:/hadoop/reducer.py" -mapper "python mapper.py" -reducer "python reducer.py" -in
put /user/geet/input/bank_data.csv\ -output /user/hadoop/output_new
```

Output:

```
C:\hadoop\bin>
C:\hadoop\bin>hadoop fs -cat
apr     56      236
aug     79      553
dec     8       11
feb     38      183
jan     16      131
jul     61      644
jun     55      475
mar     20      28
may     93      1304
nov     39      349
oct     37      42
sep     17      34

C:\hadoop\bin>
```

3. **Data Analysis with MapReduce:**

Question 3.1. Analyse the average duration of contact (in seconds) per campaign outcome ('poutcome').

Answer:

Steps:

1. **Write the MapReduce script and save as mapper.py and reducer.py**

   Note: These two code files are submitted with the document in separate files.

2. **Upload the banking data to HDFS.**
3. **Run the MapReduce job on your Hadoop cluster.**

```
C:\hadoop\bin>hadoop fs -rm -r /user/hadoop/output_new
Deleted /user/hadoop/output_new

C:\hadoop\bin>hadoop jar C:\hadoop\share\hadoop\tools\lib\hadoop-streaming-2.10.2.jar -files "file:///C:/hadoop/mapper.py,file:///C:/hadoop/reducer.py" -mapper "python mapper.py" -reducer "python reducer.py" -in
put /user/geet/input/bank_data.csv\ -output /user/hadoop/output_new
```

4. **Check and interpret the results.**
5. **Clean up the output directory if you need to rerun the job.**

Output:

```
C:\hadoop\bin>hadoop fs -cat /user/hadoop/output_new/part-00000
failure 254.38
other   273.83
success 338.64
unknown 262.10

C:\hadoop\bin>
```

Summary:

The MapReduce job analyzed the dataset to calculate the average contact duration (in seconds) for each campaign outcome. The results are as follows:

- Failure: Average contact duration is 254.38 seconds.
- Other: Average contact duration is 273.83 seconds.
- Success: Average contact duration is 338.64 seconds.
- Unknown: Average contact duration is 262.10 seconds.

These results show that successful campaign outcomes are associated with the longest average contact duration, while failed outcomes have shorter average durations.

Question 3.2 - Examine the relationship between the age of clients and their balance, and present findings in a summarised form.

Answer:

Summary:

- Purpose: This analysis directly examines how the balance varies with each specific age.
- Implementation: The MapReduce job processes the data by outputting the age and balance in the mapper and then calculates the average balance for each age in the reducer.
- Results: The results will give a detailed view of how balance varies with specific ages, providing a more granular insight into the relationship between age and balance.

This approach avoids grouping by age bins and instead provides a direct average balance for each individual age.

Steps:

1. **Write the MapReduce script and save as mapper.py and reducer.py**

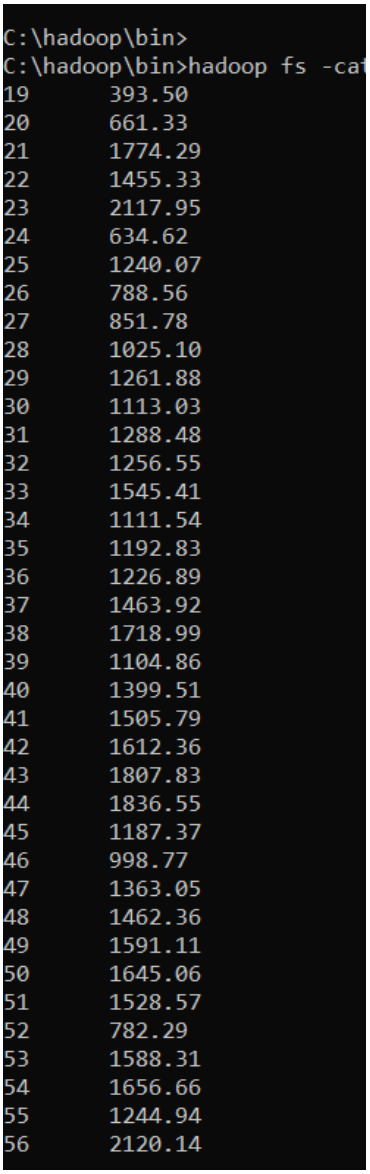
   Note: These two code files are submitted with the document in separate files.

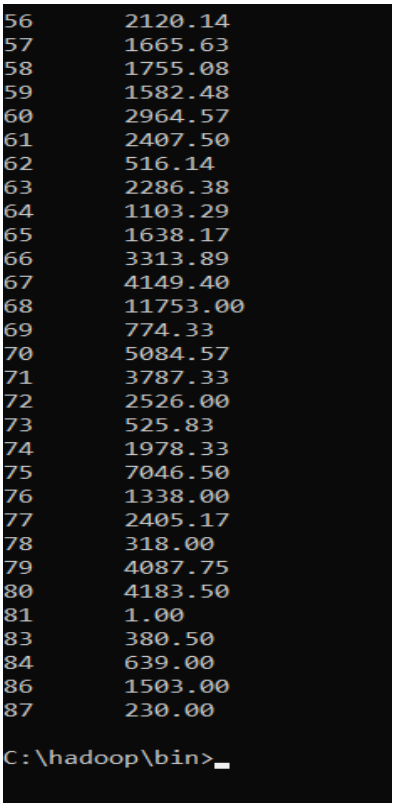
2. **Upload the banking data to HDFS.**
3. **Run the MapReduce job on your Hadoop cluster.**

```
C:\hadoop\bin>hadoop fs -rm -r /user/hadoop/output_new
Deleted /user/hadoop/output_new

C:\hadoop\bin>hadoop jar C:\hadoop\share\hadoop\tools\lib\hadoop-streaming-2.10.2.jar -files "file:///C:/hadoop/mapper.py,file:///C:/hadoop/reducer.py" -mapper "python mapper.py" -reducer "python reducer.py" -in
put /user/geet/input/bank_data.csv\ -output /user/hadoop/output_new
```

4. **Check and interpret the results.**
5. **Clean up the output directory if you need to rerun the job.**

Output:





Summary of Findings:

After running the MapReduce job, the output provides the average account balance for each specific age. Here's a summary of the key points:

- **Age-Specific Averages**: The output shows the average balance corresponding to each client's age. For example, a 23-year-old might have an average balance of 2117.95, while a 25-year-old might have an average balance of 1240.05.
- **Trends**: If observed across ages, you might notice trends such as:
    - Increase with Age: In some cases, there might be a gradual increase in average balance as age increases.

    - Fluctuations: Certain ages might show higher or lower average balances due to specific financial behaviours or life events.
- **Variability:** The average balances might fluctuate significantly across different ages, reflecting the diverse financial situations of clients at various life stages.

Conclusion:

The analysis reveals the average balance associated with each age, helping to identify any patterns or anomalies in financial behaviour across different age groups. This granular insight is valuable for financial institutions to tailor their products and services according to the needs of different age demographics.

HIVE

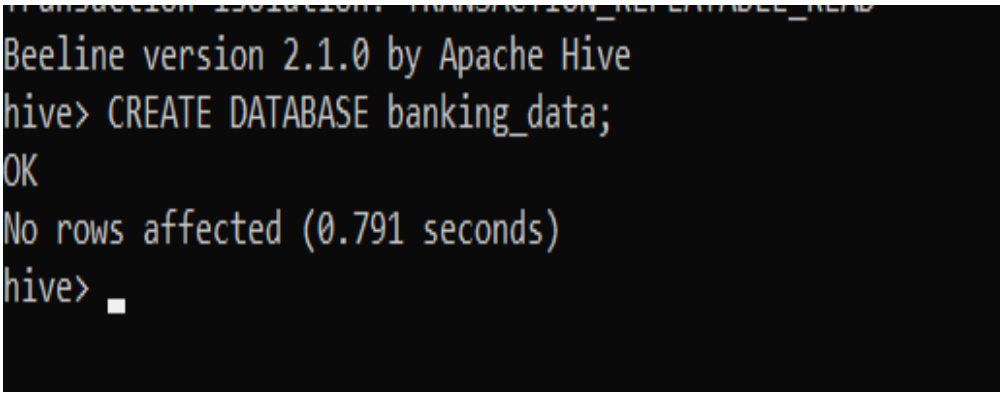
1. **Data Ingestion and Table Creation**:

Question 1.1 Create a Hive database named **banking_data**.

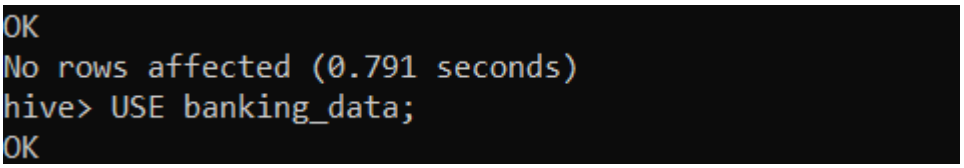
Answer:

Steps to Create a Hive Database:

1. Start Hive CLI
2. Create a new database named banking_data.



3. Switch to the newly created database:



Question 1.2. Define and create a Hive table **client_info** with appropriate data types for the **bank.csv** dataset.

Answer:

- **Create the client_info table with appropriate data types based on the columns in the bank.csv file:**

```
No rows affected (0.033 seconds)
hive> CREATE TABLE client_info (
. . > age INT,
. . > job STRING,
. . > marital STRING,
. . > education STRING,
. . > default STRING,
. . > balance INT,
. . > housing STRING,
. . > loan STRING,
. . > contact STRING,
. . > day INT,
. . > month STRING,
. . > duration INT,
. . > campaign INT,
. . > pdays INT,
. . > previous INT,
. . > poutcome STRING,
. . > y STRING
. . > )
. . > ROW FORMAT DELIMITED
. . > FIELDS TERMINATED BY ','
. . > STORED AS TEXTFILE;
OK
No rows affected (0.492 seconds)
hive>
```

Question 1.3. Load the data from the **bank.csv** file into the **client_info** table.

Answer:

1. Load the banking data from hadoop directory to the client_info table.

```
name. (state=42000,code=40000)
hive> LOAD DATA INPATH '/user/geet/input/bank_data.csv' INTO TABLE client_info;
Loading data to table banking_data.client_info
OK
No rows affected (0.67 seconds)
```

2. Query the client_info table to verify that the data has been loaded correctly.

```
No rows affected (0.67 seconds)
hive> SELECT*FROM client_info LIMIT 10;
-chgrp: 'GREAT_SUCCESS\ergee' does not match expected pattern for group
Usage: hadoop fs [generic options]
```

Output:

```
OK
 job marital education default  housing loan contact  month     poutcome y
30 unemployed married primary no 1787 no no cellular 19 oct 79 1 -1 0 unknown no
33 services married secondary no 4789 yes yes cellular 11 may 220 1 339 4 failure no
35 management single tertiary no 1350 yes no cellular 16 apr 185 1 330 1 failure no
30 management married tertiary no 1476 yes yes unknown 3 jun 199 4 -1 0 unknown no
59 blue-collar married secondary no 0 yes no unknown 5 may 226 1 -1 0 unknown no
35 management single tertiary no 747 no no cellular 23 feb 141 2 176 3 failure no
36 self-employed married tertiary no 307 yes no cellular 14 may 341 1 330 2 other no
39 technician married secondary no 147 yes no cellular 6 may 151 2 -1 0 unknown no
41 entrepreneur married tertiary no 221 yes no unknown 14 may 57 2 -1 0 unknown no
10 rows selected (0.877 seconds)
hive>
```

Here, we can see the asked ten rows in the output. So, it is verified that the data is loaded successfully.

Question 2: Basic Data Exploration:

Question 2.1. Write a HiveQL query to count the total number of clients in the dataset.

Answer:

Steps:

1. **FROM client_info**: The query selects data from the `client_info` table, which contains the records of all clients.
2. **COUNT(*) AS total_clients**: The `COUNT(*)` function counts the total number of rows in the `client_info` table. The result is labelled as `total_clients` for clarity.

Query:

```
hive> select count(*) AS total_clients FROM client_info;
-chgrp: 'GREAT_SUCCESS\ergee' does not match expected pattern for group
Usage: hadoop fs [generic options]
```

Output:

```
4522
1 row selected (2.422 seconds)
hive>
```

Summary of the Results:

- **Total Number of Clients**: The query returns a single number representing the total number of clients in the dataset. This number gives you a quick overview of the dataset size, indicating how many client records are available for analysis. So, here we can see that the total number of clients is 4522.

Question : 2.2 Display the first 10 rows of the dataset.

Answer:

Steps:

1. **FROM client_info**: The query selects data from the `client_info` table, which contains all the records in the dataset.
2. **SELECT ***: The query selects all columns (*) from the `client_info` table. This means that every piece of information available for each client will be included in the result.
3. **LIMIT 10**: The query restricts the output to only the first 10 rows of the dataset. This is useful for quickly examining a sample of the data without retrieving the entire dataset.

Query:

```
No rows affected (0.67 seconds)
hive> SELECT*FROM client_info LIMIT 10;
-chgrp: 'GREAT_SUCCESS\ergee' does not match expected pattern for group
Usage: hadoop fs [generic options]
```

Output:

```
OK
 job marital education default  housing loan contact  month     poutcome y
30 unemployed married primary no 1787 no no cellular 19 oct 79 1 -1 0 unknown no
33 services married secondary no 4789 yes yes cellular 11 may 220 1 339 4 failure no
35 management single tertiary no 1350 yes no cellular 16 apr 185 1 330 1 failure no
30 management married tertiary no 1476 yes yes unknown 3 jun 199 4 -1 0 unknown no
59 blue-collar married secondary no 0 yes no unknown 5 may 226 1 -1 0 unknown no
35 management single tertiary no 747 no no cellular 23 feb 141 2 176 3 failure no
36 self-employed married tertiary no 307 yes no cellular 14 may 341 1 330 2 other no
39 technician married secondary no 147 yes no cellular 6 may 151 2 -1 0 unknown no
41 entrepreneur married tertiary no 221 yes no unknown 14 may 57 2 -1 0 unknown no
10 rows selected (0.877 seconds)
hive>
```

Summary of the Results:

- **First 10 Rows of the Dataset**: Here, we can see that the output displays all columns and their values for the first 10 clients in the `client_info` table. These rows represent a small sample of the overall dataset, providing a snapshot of the data structure and contents.

Question 3. Data Filtering and Sorting:

Question 3.1 Retrieve all records of clients who are married and have a personal loan.

Answer:

Steps:

1. **FROM client_info**: The query selects data from the `client_info` table, which includes various details about the clients such as their marital status, loan status, and other attributes.
2. **SELECT ***: The query selects all columns (*) from the `client_info` table. This means that every piece of information available for each client will be included in the result, such as age, job, balance, etc.
3. **WHERE marital = 'married' AND loan = 'yes'**: The query filters the data to include only those clients who meet both of the following conditions:

- **marital = 'married'**: The client must be married.
- **loan = 'yes'**: The client must have a personal loan.

Query:

```
1 row selected (2.422 seconds)
hive> select * from client_info where marital ='married'AND loan ='yes';
-chgrp: 'GREAT_SUCCESS\ergee' does not match expected pattern for group
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
```

Output:

```
Usage: hadoop fs [generic options] -chgrp [-R] GROUP PATH...
OK
33 services married secondary no 4789 yes yes cellular 11 may 220 1 339 4 failure no
30 management married tertiary no 1476 yes yes unknown 3 jun 199 4 -1 0 unknown no
43 services married primary no -88 yes yes cellular 17 apr 313 1 147 2 failure no
31 blue-collar married secondary no 360 yes yes cellular 29 jan 89 1 241 1 failure no
40 management married tertiary no 194 no yes cellular 29 aug 189 2 -1 0 unknown no
56 self-employed married secondary no 784 no yes cellular 30 jul 149 2 -1 0 unknown no
53 admin. married secondary no 105 no yes cellular 21 aug 74 2 -1 0 unknown no
57 management married secondary no 82 no yes telephone 4 feb 140 1 -1 0 unknown no
41 blue-collar married primary no -516 no yes telephone 8 jul 554 3 -1 0 unknown no
41 management married secondary no 0 no yes cellular 7 jul 630 3 -1 0 unknown no
37 blue-collar married secondary no 427 yes yes unknown 9 jun 371 3 -1 0 unknown no
32 self-employed married secondary no 217 yes yes cellular 15 jul 317 5 -1 0 unknown no
36 blue-collar married secondary no -231 no yes cellular 15 jul 779 2 -1 0 unknown no
42 admin. married secondary no 323 yes yes unknown 8 may 280 2 -1 0 unknown no
35 management married tertiary no 106 no yes cellular 11 aug 588 2 -1 0 unknown no
56 retired married primary no 1906 no yes unknown 19 jun 45 9 -1 0 unknown no
43 services married secondary no 978 yes yes unknown 26 may 82 2 -1 0 unknown no
43 admin. married secondary no -465 yes yes cellular 23 jul 166 1 -1 0 unknown no
44 admin. married secondary no 5181 yes yes cellular 31 jul 18 7 -1 0 unknown no
49 blue-collar married primary no 0 yes yes telephone 23 jul 97 6 -1 0 unknown no
38 services married secondary no 1 no yes cellular 21 nov 152 2 -1 0 unknown no
42 technician married secondary no 2030 yes yes cellular 9 jul 196 1 -1 0 unknown no
49 blue-collar married primary no 305 yes yes telephone 10 jul 834 10 -1 0 unknown no
35 technician married tertiary no 0 yes yes cellular 23 sep 112 1 62 6 other no
27 admin. married secondary no -247 yes yes unknown 4 jun 344 2 -1 0 unknown no
43 technician married secondary no 0 no yes cellular 8 may 9 2 172 5 failure no
55 blue-collar married secondary no 989 yes yes unknown 23 may 246 4 -1 0 unknown no
34 management married tertiary no 415 no yes cellular 23 jul 361 2 -1 0 unknown no
54 housemaid married secondary no 209 yes yes cellular 25 jul 97 1 -1 0 unknown no
53 entrepreneur married tertiary no 624 no yes cellular 21 jul 180 4 -1 0 unknown no
40 self-employed married secondary no 1516 yes yes unknown 23 may 373 1 -1 0 unknown no
33 technician married tertiary no -988 yes yes cellular 15 jul 83 3 -1 0 unknown no
34 admin. married tertiary no 69 no yes cellular 4 aug 120 3 1 6 success yes
38 entrepreneur married secondary no 593 yes yes cellular 24 jul 1484 24 -1 0 unknown yes
56 management married unknown no 353 no yes cellular 25 jul 171 2 -1 0 unknown no
56 technician married secondary no 205 no yes cellular 23 jul 442 2 -1 0 unknown no
38 unemployed married primary no 1147 yes yes unknown 8 may 249 5 -1 0 unknown no
49 blue-collar married secondary no 8545 yes yes cellular 6 may 199 7 167 1 failure no
43 management married tertiary no 2 no yes cellular 20 aug 472 2 -1 0 unknown no
50 admin. married primary no 276 no yes unknown 17 jun 641 6 -1 0 unknown no
47 blue-collar married primary no 214 yes yes unknown 9 jun 168 1 -1 0 unknown no
51 unemployed married secondary no 1760 no yes cellular 19 nov 162 1 -1 0 unknown no
41 management married tertiary no 832 yes yes cellular 21 nov 164 1 176 1 failure no
44 admin. married secondary no 0 yes yes unknown 16 may 159 1 -1 0 unknown no
39 management married tertiary no 278 no yes cellular 28 jul 60 3 -1 0 unknown no
42 blue-collar married primary no 0 yes yes cellular 7 may 209 2 -1 0 unknown no
38 blue-collar married primary no 168 yes yes telephone 13 may 212 3 -1 0 unknown no
```

```
51 management married tertiary no 176 yes yes unknown 1 jun 8 1 687 2 failure no
45 unemployed married primary no 180 yes yes unknown 27 may 62 2 -1 0 unknown no
26 entrepreneur married tertiary no 37 yes yes cellular 6 may 207 1 363 2 failure no
46 technician married secondary no 410 no yes cellular 21 nov 162 5 -1 0 unknown no
40 blue-collar married primary no 177 yes yes telephone 21 jul 742 4 -1 0 unknown no
30 technician married secondary no -393 yes yes cellular 16 jul 435 1 -1 0 unknown yes
33 technician married secondary no -163 yes yes cellular 8 aug 502 2 -1 0 unknown no
29 blue-collar married secondary no 424 yes yes unknown 14 may 241 2 -1 0 unknown no
30 services married secondary no 227 no yes unknown 3 jun 23 1 -1 0 unknown no
56 self-employed married secondary no 800 no yes cellular 9 jul 170 1 -1 0 unknown no
38 blue-collar married primary no 506 yes yes unknown 12 jun 122 1 -1 0 unknown no
33 services married secondary no 210 yes yes unknown 20 may 201 1 -1 0 unknown no
36 unemployed married secondary no -872 yes yes cellular 20 nov 153 1 183 1 failure no
38 services married secondary no 0 no yes cellular 16 jul 1473 6 -1 0 unknown no
48 management married tertiary no 5057 no yes cellular 19 nov 37 1 -1 0 unknown no
37 management married tertiary no 4039 no yes cellular 25 jul 106 2 -1 0 unknown no
34 management married tertiary no 997 yes yes cellular 21 nov 81 1 -1 0 unknown no
31 technician married secondary no 0 no yes cellular 15 jul 624 6 -1 0 unknown no
32 services married secondary yes -220 yes yes cellular 25 jul 123 2 -1 0 unknown no
58 retired married secondary no 3382 no yes cellular 28 may 294 2 309 2 failure no
50 technician married tertiary no 199 yes yes cellular 3 feb 116 2 253 1 failure no
35 admin. married secondary no 147 yes yes cellular 29 jan 41 1 184 7 failure no
54 technician married secondary no 2225 no yes cellular 13 aug 73 2 -1 0 unknown no
31 housemaid married unknown yes -6 no yes telephone 7 jul 94 2 -1 0 unknown no
50 technician married tertiary no 3337 yes yes telephone 31 jul 24 14 -1 0 unknown no
36 services married secondary no 895 yes yes unknown 4 jun 622 1 -1 0 unknown no
37 services married secondary no 0 yes yes cellular 31 jul 187 2 -1 0 unknown no
58 blue-collar married secondary no -27 no yes telephone 31 jul 77 13 -1 0 unknown no
41 blue-collar married primary no 293 yes yes cellular 7 may 102 1 -1 0 unknown no
50 management married tertiary no 19447 yes yes cellular 21 nov 166 1 -1 0 unknown no
32 admin. married secondary no 0 yes yes cellular 17 nov 159 2 195 2 failure no
57 blue-collar married primary no 5431 yes yes unknown 6 may 383 1 -1 0 unknown no
28 blue-collar married secondary no 225 yes yes unknown 7 may 866 2 -1 0 unknown no
31 blue-collar married secondary no 3653 yes yes cellular 21 nov 252 1 168 4 failure no
34 management married tertiary no 436 no yes cellular 28 jul 118 4 -1 0 unknown no
30 technician married tertiary no 101 yes yes cellular 7 jul 187 3 -1 0 unknown no
33 unemployed married tertiary no 302 yes yes cellular 16 apr 670 1 -1 0 unknown no
34 management married tertiary no 1557 yes yes unknown 13 may 213 1 -1 0 unknown no
35 services married secondary no 505 yes yes unknown 27 may 371 2 -1 0 unknown no
37 blue-collar married primary no 190 yes yes unknown 8 may 194 1 -1 0 unknown no
40 management married tertiary no -17 yes yes cellular 11 may 474 1 256 1 success yes
55 self-employed married secondary no 2678 no yes cellular 18 aug 151 12 -1 0 unknown no
42 services married secondary no -91 yes yes cellular 5 feb 43 1 -1 0 unknown no
60 self-employed married primary no 362 no yes cellular 29 jul 816 6 -1 0 unknown yes
42 admin. married unknown no 642 yes yes unknown 16 may 509 2 -1 0 unknown no
57 self-employed married tertiary yes -3313 yes yes unknown 9 may 153 1 -1 0 unknown no
453 rows selected (0.856 seconds)
hive>
```

Summary of result:

- This query filters the data to retrieve records of clients who are married and have a personal loan.
- **Output**: We can see the output is a list of all married clients who have taken a personal loan, including all columns of data.

Question 3.2. List the top 10 clients with the highest balance, displaying their job, marital status, and balance.

Answer:

Steps:

1. **FROM client_info**: The query selects data from the `client_info` table, which contains information about clients, including their job, marital status, and account balance.
2. **SELECT job, marital, balance**: The query specifies that it wants to retrieve the `job`, `marital`, and `balance` columns from the `client_info` table.
3. **ORDER BY balance DESC**: The query orders the results by the `balance` column in descending order (DESC), meaning that the clients with the highest balances will appear first.
4. **LIMIT 10**: The query limits the results to the top 10 records. This means only the 10 clients with the highest balances will be shown.

Query:

```
hive> select job,marital,balance from client_info order by balance desc limit 10;
-chgrp: 'GREAT_SUCCESS\ergee' does not match expected pattern for group
Usage: hadoop fs [generic options]
```

Result:

```
retired married 71188
entrepreneur married 42045
technician single 27733
management married 27359
technician married 27069
housemaid single 26965
retired married 26452
services married 26394
management divorced 26306
retired single 25824
10 rows selected (1.677 seconds)
hive>
```

Summary of the Results:

Top 10 Clients by Balance: The output of this query will display the job, marital status, and balance of the 10 clients who have the highest account balances. So, here we see that **retired clients** are generally married and have the highest balance.

Question 4.: Data Aggregation and Grouping:

Question 4.1 Calculate the average age of clients for each job category.

Steps:

The query retrieves data from the `client_info` table, which contains information about clients, including their job and age.

GROUP BY job: The query groups the data by the `job` column. This means that the data will be aggregated separately for each unique job category.

AVG(age) AS average_age: For each job category, the query calculates the average age of the clients using the `AVG(age)` function. The result is stored in a column named `average_age`.

SELECT job, AVG(age) AS average_age: Finally, the query selects and displays the job category (`job`) alongside the calculated average age (`average_age`) for each group.

Query:

```
hive> select job, AVG(age) AS average_age from client_info group by job;
-chgrp: 'GREAT_SUCCESS\ergee' does not match expected pattern for group
```

Result:

```
WARNING: Hive-on-MR is deprecated in Hive 2 and may
admin. 39.68200836820084
blue-collar 40.15644820295983
entrepreneur 42.01190476190476
housemaid 47.339285714285715
job
management 40.54076367389061
retired 61.869565217391305
self-employed 41.45355191256831
services 38.57074340527578
student 26.821428571428573
technician 39.470052083333336
unemployed 40.90625
unknown 48.10526315789474
13 rows selected (1.643 seconds)
hive>
```

Summary of the Results:

- **Average Age by Job Category**: The query outputs the average age of clients for each job category. This provides insight into the typical age of clients in different professions. **Here, we can see that the average age of the majority of clients for different job categories is between 35 to 45.**

Question. 4.2. Find the total number of clients for each education level who have defaulted on credit.

Answer:

Steps:

FROM client_info: The query starts by selecting data from the `client_info` table, which contains information about the clients, including their education level and whether they have defaulted on credit.

WHERE default = 'yes': The query filters the data to include only those clients who have defaulted on credit. The `default` column is checked, and only records where `default = 'yes'` are selected. This ensures that the query is only counting clients who have actually defaulted.

GROUP BY education and default: The query then groups the filtered data by the `education` column and default column.

COUNT(*) AS total_defaulted_clients: For each education level, the query counts the number of clients who have defaulted on credit using `COUNT(*)`. The result is stored in a column named `total_defaulted_clients`.

SELECT education, default, COUNT(*) AS total_defaulted_clients: Finally, the query selects and displays the education level alongside the total number of clients who have defaulted in that education level.

Query:

```
13 rows selected (1.643 seconds)
hive> select education, default, count(*) AS total_defaulted_client from client_info where default = 'yes' group by education,default;
-chgrp: 'GREAT_SUCCESS\ergee' does not match expected pattern for group
Usage: hadoop fs [generic options]
```

Output:

```
WARNING: Hive-on-MR is deprecated
primary yes 10
secondary yes 46
tertiary yes 17
unknown yes 3
4 rows selected (1.726 seconds)
hive>
```

Summary of result:

- This query finds the total number of clients for each education level who have defaulted on credit. Here, we can see that the majority of clients who defaulted have a secondary education level.

Question: 5 Complex Queries for Insights:

Question 5.1. Identify the top 5 job categories with the highest average balance and the percentage of clients in each of these job categories who have subscribed to a term deposit.

Answer:

Step 1: Inner Query - Identifying Top 5 Job Categories by Average Balance

1. **FROM client_info**: The query selects data from the `client_info` table.
2. **GROUP BY job**: It groups the data by the `job` column, meaning it will aggregate data for each job category.
3. **AVG(balance) AS avg_balance**: For each job category, the query calculates the average balance (`AVG(balance)`) and stores it as `avg_balance`.
4. **COUNT(*) AS total_clients**: It counts the total number of clients in each job category using `COUNT(*)` and stores this value as `total_clients`.
5. **SUM(CASE WHEN y = 'yes' THEN 1 ELSE 0 END) AS subscribed_clients**: It counts the number of clients who subscribed to a term deposit (`y = 'yes'`) within each job category. This is done using a `CASE` statement, where `1` is added for each subscription and `0` for non-subscriptions. The result is stored as `subscribed_clients`.
6. **ORDER BY avg_balance DESC**: The job categories are ordered by their average balance in descending order, so the job categories with the highest average balance come first.
7. **LIMIT 5**: The query limits the result to the top 5 job categories with the highest average balance.

Step 2: Outer Query - Calculating Subscription Percentage

1. **FROM (...) sc**: The outer query selects data from the result of the inner query, which is aliased as `sc`.
2. **SELECT sc.job, sc.avg_balance**: The outer query directly selects the `job` and `avg_balance` columns from the inner query's results.
3. **(sc.subscribed_clients / sc.total_clients) * 100 AS subscription_percentage**: It calculates the subscription percentage by dividing the number of subscribed clients (`subscribed_clients`) by the total number of clients (`total_clients`) for each job category, then multiplying by 100 to express it as a percentage. This value is stored as `subscription_percentage`.

Query:

```
hive> select sc.job, sc.avg_balance, (sc.subscribed_clients / sc.total_clients) * 100 as subscription_percentage from (select job, AVG(balance) as avg_balance, count(*) as total_clients, sum(case when y = 'yes' t
hen 1 else 0 end) as subscribed_clients from client_info group by job order by avg_balance desc limit 5) sc;
-chgrp: 'GREAT_SUCCESS\ergee' does not match expected pattern for group
Usage: hadoop fs [generic options]
        [-appendToFile <localsrc> ... <dst>]
```

Output:

```
retired 2319.191304347826 23.47826086956522
housemaid 2083.8035714285716 12.5
management 1766.9287925696594 13.519091847265221
entrepreneur 1645.125 8.928571428571429
student 1543.8214285714287 22.61904761904762
5 rows selected (2.922 seconds)
hive>
```

Summary of the Results:

- **Top 5 Job Categories by Average Balance**: The query identifies the five job categories with the highest average balances. These are the clients with the most significant average account balances across all job types. So, here retired and housemaids have the highest average balance.
- **Subscription Percentage**: The query then calculates how successful the current campaign was in converting clients in these high-balance job categories into term deposit subscribers. The `subscription_percentage` indicates the effectiveness of the campaign for each job category. E.g. Here it was not very successful for entrepreneurs because the subscription_rate was only 8.928 percent.

Question 5.2. Determine the month with the highest number of contacts and the success rate of the campaign in that month (percentage of clients who subscribed to a term deposit).

Answer:

Steps:

Inner query:

1. **FROM client_info**: The query starts by selecting data from the `client_info` table.
2. **GROUP BY month**: It groups the data by the `month` column to calculate statistics for each month.
3. **COUNT(*) AS total_contacts**: For each month, it counts the total number of records (or contacts) using `COUNT(*)`. This value is stored as `total_contacts`.
4. **SUM(CASE WHEN y = 'yes' THEN 1 ELSE 0 END) AS successful_contacts**: For each month, the query counts the number of records where `y = 'yes'` (i.e., the client subscribed to a term deposit). This is done using a `CASE` statement that returns `1` for successful contacts and `0` otherwise. The sum of these values gives the number of successful contacts, stored as `successful_contacts`.
5. **ORDER BY total_contacts DESC**: The query orders the results by `total_contacts` in descending order, meaning the month with the highest number of contacts will appear first.
6. **LIMIT 1**: Finally, it limits the results to just the top month (the month with the highest number of contacts).

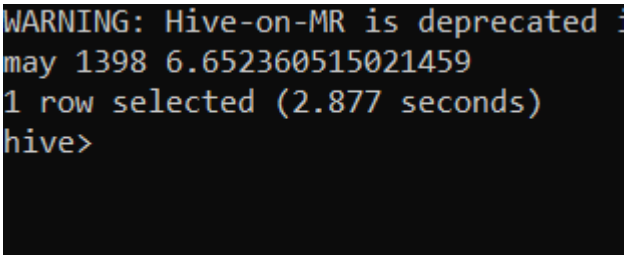
Outer Query:

1. **FROM (...) AS top_month: The outer query selects data from the result of the inner query, which has been aliased as `top_month`.**
2. **SELECT month, total_contacts: The outer query selects the `month` and `total_contacts` fields directly from the result of the inner query.**
3. **(successful_contacts / total_contacts) * 100 AS success_rate: It calculates the `success_rate` by dividing the number of `successful_contacts` by `total_contacts` and then multiplying by 100 to get a percentage.**

Query:

```
student 1543.8214285714287 22.61904761904762
5 rows selected (2.922 seconds)
hive> select month, total_contacts, (successful_contacts/total_contacts) * 100 as success_rate from (select month, count(*) as total_contacts, sum(case when y = "yes" then 1 else 0 end) as successful_contacts fr
om client_info group by month order by total_contacts DESC limit 1) as top_month;
-chgrp: 'GREAT_SUCCESS\ergee' does not match expected pattern for group
Usage: hadoop fs [generic options]
```

Output:



Summary:

- This query determines the month with the highest number of client contacts and calculates the success rate of the campaign in that month (the percentage of clients who subscribed to a term deposit).
- **Output**: May, is the month with the highest contacts, the total number of contacts are 1398, and the success rate (percentage of clients who subscribed to a term deposit) during that month is 6.652.

Question .6. Correlation Analysis:

Question .6. Calculate the correlation between age and balance for the clients.

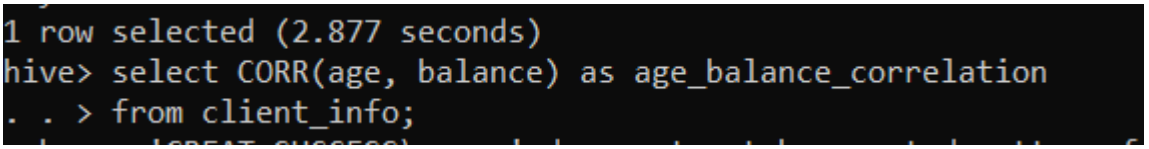
Answer:

The `CORR(age, balance)` function calculates the Pearson correlation coefficient between the `age` and `balance` columns.

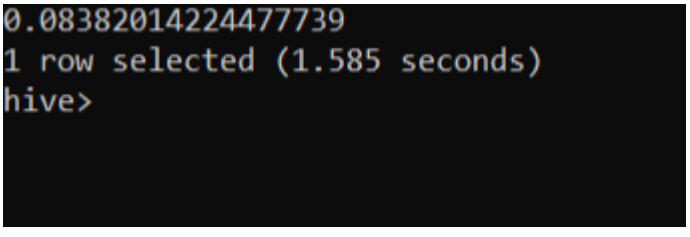
This coefficient will range from -1 to 1, where:

- **1** indicates a perfect positive correlation,
- **-1** indicates a perfect negative correlation, and
- **0** indicates no correlation.

Query:



Output:



Summary Of Result:

This result gives you an idea of how strongly the age of clients is related to their account balance. Here, Since the output is 0.0838 i.e. between 0 and 1 so, we can say that age is slightly related to balance.

Question. 7. Trend Analysis:

Analyse the year-over-year trend in the number of clients contacted:

***There is no data in the bank_data.csv dataset which represents the year. But let's say the first four characters from the `month` column represent the year (e.g., `2023` from `2023-Jan`).**

Answer:

Steps:

1. Extract Year: The `SUBSTRING(month, 1, 4)` function extracts the first four characters from the `month` column, assuming they represent the year (e.g., `2023` from `2023-Jan`).
2. Count Contacts by Year: We group the data by the extracted year and count the number of clients contacted in each year.

Query:

```
1 row selected (1.585 seconds)
hive> SELECT SUBSTRING(month,1,4) AS year,
. . > COUNT(*) AS num_clients_contacted
. . > FROM
. . > CLIENT_INFO
. . > GROUP BY SUBSTRING(month,1,4)
. . > ORDER BY year;
-chgrp: 'GREAT_SUCCESS\ergee' does not match expected pattern for group
Usage: hadoop fs [generic options]
```

Output:

```
apr 293
aug 633
dec 20
feb 222
jan 148
jul 706
jun 531
mar 49
may 1398
mont 1
nov 389
oct 80
sep 52
13 rows selected
hive>
```

Summary of result: This output will give us a clear view of the year-over-year trend in the number of client contacts. Here, we did not have year information in the bank_data.csv file. So, we could not get the desired result.

Question.8.Anomaly Detection:

Identify any unusual patterns in the average yearly balance across different education levels.

Answer:

Inner Query (Subquery):

- **`SUBSTRING(month, 1, 4) AS year`**: Extracts the year from the `month` column, assuming `month` is in a format that includes the year (e.g., `2024-08`).
- **`education`**: Selects the `education` column from the `client_info` table.
- **`AVG(balance) AS avg_yearly_balance`**: Calculates the average balance for clients grouped by year and education.
- **`AVG(AVG(balance)) OVER (PARTITION BY SUBSTRING(month, 1, 4)) AS overall_avg_balance`**: Calculates the overall average balance for each year across all education levels using a window function.
- **`STDDEV(AVG(balance)) OVER (PARTITION BY SUBSTRING(month, 1, 4)) AS stddev_balance`**: Calculates the standard deviation of the average balance for each year across all education levels using a window function.
- **`GROUP BY SUBSTRING(month, 1, 4), education`**: Groups the data by year and education to calculate the metrics above.

Outer Query:

- **`(avg_yearly_balance - overall_avg_balance) / stddev_balance AS z_score`**: Calculates the z-score for the average yearly balance. The z-score measures how many standard deviations a data point is from the mean. In this context, it tells us how much the average balance for a specific year and education level deviates from the overall average balance for that year.

Query:

```
hive> select year, education, (avg_yearly_balance - overall_avg_balance) / stddev_balance AS z_score
. . > FROM (
. . > SELECT
. . > SUBSTRING(month, 1, 4) AS year,
. . > education,
. . > AVG(balance) AS avg_yearly_balance,
. . > AVG(AVG(balance)) OVER (PARTITION BY SUBSTRING(month, 1, 4)) AS overall_avg_balance,
. . > STDDEV(AVG(balance)) OVER (PARTITION BY SUBSTRING(month, 1, 4)) AS stddev_balance
. . > FROM client_info
. . > GROUP BY SUBSTRING(month, 1, 4), education
. . > ) AS subquery;
-chgrp: 'GREAT_SUCCESS\ergee' does not match expected pattern for group
Usage: hadoop fs [generic options]
```

Output:

```
OK
WARNING: Hive-on-MR is deprecated in Hi
apr primary 1.267794378462834
apr secondary -0.7235770026264813
apr tertiary 0.6555354947097891
apr unknown -1.19975287054614l3
aug primary -0.8113812295586758
aug secondary -0.57757328901129l9
aug tertiary -0.31633238372712924
aug unknown 1.7052869022970998
dec primary -0.8088635843923103
dec secondary -0.3975683783904l1
dec tertiary -0.5058241544774784
dec unknown 1.7122561172602497
feb primary -1.0701923004119014
feb secondary -0.9216091477916096
feb tertiary 0.8917649610011248
feb unknown 1.10003648720238l2
jan primary 1.1404156384061253
jan secondary -0.4369029752604172
jan tertiary 0.7115176049750513
jan unknown -1.415030268120l6
jul primary 0.9649291030842655
jul secondary -0.42584562596808l4
jul tertiary 0.901411256061034l
jul unknown -1.4404947331772213
jun primary -0.156827791766341l6
jun secondary -1.0365919570822748
jun tertiary 1.64283529945017l7
jun unknown -0.44941555060155l86
mar primary -1.072287607352997l
mar secondary -0.24628248100747l7
mar tertiary 1.6391398006892093
mar unknown -0.3205697123287344
may primary -1.00482486824628l6
may secondary -0.9457054758622359
may tertiary 0.66407083933434l4
may unknown 1.28645950477416l1
mont education
nov primary -0.8137802862217836
nov secondary 0.2770762379753206
nov tertiary 1.5167439470393023
nov unknown -0.9800398987928378
oct primary 1.709788975533738l
oct secondary -0.370527874773251l4
oct tertiary -0.52382141974328l3
oct unknown -0.8154396810172013
sep primary -1.3086820436917062
```

```
dec unknown 1.7122561172602497
feb primary -1.0701923004119014
feb secondary -0.9216091477916096
feb tertiary 0.8917649610011248
feb unknown 1.1000364872023862
jan primary 1.1404156384061253
jan secondary -0.4369029752604172
jan tertiary 0.7115176049750513
jan unknown -1.41503026812076
jul primary 0.9649291030842655
jul secondary -0.4258456259680814
jul tertiary 0.9014112560610348
jul unknown -1.4404947331772213
jun primary -0.15682779176634196
jun secondary -1.0365919570822748
jun tertiary 1.6428352994501707
jun unknown -0.44941555060155386
mar primary -1.0722876073529977
mar secondary -0.2462824810074777
mar tertiary 1.6391398006892093
mar unknown -0.3205697123287344
may primary -1.0048248682462826
may secondary -0.9457054758622359
may tertiary 0.6640708393343474
may unknown 1.2864595047741691
mont education
nov primary -0.8137802862217836
nov secondary 0.2770762379753206
nov tertiary 1.5167439470393023
nov unknown -0.9800398987928378
oct primary 1.7097889755337383
oct secondary -0.37052787477325194
oct tertiary -0.5238214197432853
oct unknown -0.8154396810172013
sep primary -1.3086820436917062
sep secondary -0.4707154583719365
sep tertiary 0.3984500713422833
sep unknown 1.38094743072136
49 rows selected (1.701 seconds)
hive>
```

Summary of the Results:

- **`year`: The year extracted from the `month` column.**
- **`education`: The education level of the clients.**
- **`z_score`: The z-score indicates how the average yearly balance for a specific education level compares to the overall average for that year. A positive z-score means the average balance is above the overall average, while a negative z-score indicates it is below the overall average.**

Output Interpretation:

- **Positive Z-Scores: Indicate that the clients with a specific education level have a higher-than-average balance compared to others in the same year.**
- **Negative Z-Scores: Indicate that the clients with a specific education level have a lower-than-average balance compared to others in the same year.**

Question . 9. Advanced Analysis:

Question 9.1. Analyze the impact of previous campaign outcomes (**poutcome**) on the current campaign's success. Calculate the subscription rate (to term deposits) for each **poutcome** category.

Answer:

Select Columns:

- **`poutcome`**: This column indicates the outcome of the previous marketing campaign (e.g., 'success', 'failure', etc.).
- **`count(*) AS total_clients`**: Counts the total number of clients for each `poutcome` category.
- **`sum(case when y = 'yes' then 1 else 0 end) as subscribed_clients`**: Counts the number of clients who subscribed to the term deposit (`y = 'yes'`) for each `poutcome` category.
- **`ROUND(SUM(case when y ='yes' then 1 else 0 end) * 100.0 / COUNT(*), 2) AS subscription_rate`**: Calculates the subscription rate as a percentage and rounds it to two decimal places. This rate is the proportion of clients who subscribed to the term deposit out of the total clients in each `poutcome` category.

From:

- The query uses the `client_info` table, which contains the relevant data.

Group By:

- Groups the results by the `poutcome` column, meaning that the calculations (total clients, subscribed clients, and subscription rate) are done separately for each distinct value of `poutcome`.

Order By:

- Orders the results by `subscription_rate` in descending order, showing the outcomes with the highest subscription rates first.

Query:

```
hive> select
. . > poutcome,
. . > count(*) AS total_clients,
. . > sum(case when y = 'yes' then 1 else 0 end) as subscribed_clients,
. . > ROUND(SUM(case when y ='yes' then 1 else 0 end) * 100.0 / COUNT(*), 2) AS subscription_rate
. . > FROM
. . > client_info
. . > GROUP BY
. . > poutcome
. . > ORDER BY
. . > subscription_rate DESC;
```

Result:

```
success 129 83 64.34
other 197 38 19.29
failure 490 63 12.86
unknown 3705 337 9.1
poutcome 1 0 0.0
5 rows selected (2.943 seconds)
hive>
```

Summary of the Results:

- **`poutcome`**: Represents different outcomes of the previous marketing campaigns, such as success or failure.
- **`total_clients`**: The total number of clients associated with each outcome.
- **`subscribed_clients`**: The number of clients who subscribed to the term deposit within each `poutcome` category.
- **`subscription_rate`**: The percentage of clients who subscribed to the term deposit, calculated per `poutcome` category.

The results of this query help to understand the effectiveness of previous marketing outcomes (`poutcome`) by showing how many clients subscribed to the term deposit in each category. By sorting the results by `subscription_rate`, it highlights which previous outcomes were most successful in leading to subscriptions.

Interpretation:

- **High Subscription Rate:** A high subscription rate in poutcome category (e.g., `success`) indicates that clients who had a positive outcome in the previous campaign are more likely to subscribe again in the current campaign.
- **Low Subscription Rate:** A low subscription rate (e.g., `failure` or `unknown`) suggests that clients with unsuccessful or unknown outcomes in the previous campaign are less likely to subscribe in the current campaign.

Here, we can see that the subscription_rate is very high i.e. 64.34 for success poutcome. So, we can say that clients who had a positive outcome in the previous campaign are more likely to subscribe again in the current campaign.

Question 9.2. Compare the average contact duration for clients who subscribed and who did not subscribe to a term deposit.

Answer:

Select Columns:

- **`y as subscription_status`**: The `y` column is renamed as `subscription_status`. This column indicates whether the client subscribed to a term deposit (`yes` or `no`).
- **`AVG(duration) AS avg_contact_duration`**: Calculates the average duration of the contact in seconds for each `subscription_status`. The `duration` column represents the last contact duration with the client.

From:

- The data is retrieved from the `client_info` table.

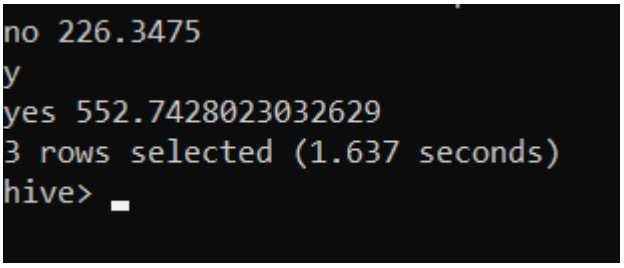
Group By:

- The results are grouped by the `subscription_status`, meaning the average contact duration is calculated separately for clients who subscribed (`y = 'yes'`) and those who did not (`y = 'no'`).

Query:

```
5 rows selected (2.943 seconds)
hive> select
. . > y as subscription_status,
. . > AVG(duration) AS avg_contact_duration
. . > FROM
. . > client_info
. . > group by
. . > y;
-chgrp: 'GREAT_SUCCESS\ergee' does not match expected pattern for group
Usage: hadoop fs [generic options]
```

Result:



Summary of the Results:

- **`subscription_status`**: This indicates whether the client subscribed to the term deposit (`yes` or `no`).
- **`avg_contact_duration`**: This is the average duration (in seconds) of contact with the client, grouped by whether they subscribed or not.

The result provides insight into whether there is a difference in the average contact duration between clients who subscribed to the term deposit and those who did not. For instance, here higher average contact duration for the `yes` group suggests that longer interactions are more effective in convincing clients to subscribe.

Submission Guidelines:

- Make a copy of this doc file.
- Perform the analysis in your local system using Hadoop and Hive and provide screenshots of both the **code** and the **output** under each question.
- Upload the doc file with other files and submit it in the submission dashboard.