***Hadoop***

1. Data Ingestion:

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

Steps:

1. Access the Hadoop Environment

* Set up the Hadoop environment
* We did this using local Hadoop as well as using Hadoop in a Docker environment.
* 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.

A screenshot of a computer

AI-generated content may be incorrect.

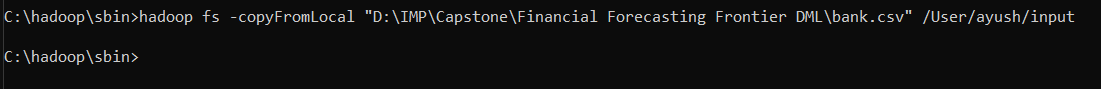
* Create a Directory in HDFS:

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



* Transfer the Banking Dataset from Local to HDFS:

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



* Check if data is loaded to the new directory:

1. We can visualise the data in our localhost by accessing localhost:9870 in our browser

A screenshot of a computer

AI-generated content may be incorrect.

1. 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:

* 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

A screen shot of a computer

AI-generated content may be incorrect.

* Access the Container Shell:

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

Docker exec -it namenode bash



* Create an input folder:

1. Inside the namenode container use hadoop fs -mkdir command to create a new directory.



* 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.

A black screen with white text

AI-generated content may be incorrect.

* Check if data is loaded to the new directory:

1. We can visualise the data in our localhost by accessing to localhost:9870 in our browser.

A screenshot of a computer

AI-generated content may be incorrect.

1. Data Transformation with MapReduce:

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

Steps:

1. Write the MapReduce Python script and save as Mapper.py and Reducer.py.
2. Create a new directory and upload the banking dataset bank.csv from local to HDFS.
3. Run Your MapReduce Job by specifying the input and output paths.
4. Retrieve the results from the output directory
5. Use hadoop fs -cat /user/hadoop/output\_new/part-00000 command to visualise and retrieve the output from the output directory

A screenshot of a computer

AI-generated content may be incorrect.

Output:

A computer screen shot of a computer program

AI-generated content may be incorrect.

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

Steps:

1. Write the MapReduce Python script and save as Mapper.py and Reducer.py.
2. . Run Your MapReduce Job by specifying the input and output paths.

A black screen with white text

AI-generated content may be incorrect.

A black screen with white text

AI-generated content may be incorrect.

Output:

A screen shot of a computer

AI-generated content may be incorrect.

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

Steps:

1. Delete the output directory before running a new MapReduce job.
2. Write the MapReduce Python script and save as Mapper.py and Reducer.py.
3. Run Your MapReduce Job by specifying the input and output paths.

A black background with white text

AI-generated content may be incorrect.

Output:

A screenshot of a computer

AI-generated content may be incorrect.

1. Data Analysis with MapReduce:

* Analyze the average duration of contact (in seconds) per campaign outcome ('poutcome').

Steps:

1. Write the MapReduce script and save as mapper.py and reducer.py
2. Upload the banking data to HDFS.
3. Run the MapReduce job on your Hadoop cluster.

A black background with white text

AI-generated content may be incorrect.

1. Check and interpret the results.
2. Clean up the output directory if you need to rerun the job.

Output:

A black screen with white text

AI-generated content may be incorrect.

Summary:

The MapReduce job analysed 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.

* Examine the relationship between the age of clients and their balance, and present findings in a summarized form.

1. Purpose: This analysis directly examines how the balance varies with each specific age.
2. 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.
3. 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
2. Upload the banking data to HDFS.
3. Run the MapReduce job on your Hadoop cluster.

A black screen with white text

AI-generated content may be incorrect.

1. Check and interpret the results.
2. Clean up the output directory if you need to rerun the job.

Output:

A screen shot of a computer

AI-generated content may be incorrect. A screenshot of a computer screen

AI-generated content may be incorrect.

*Summary of Findings:*

After executing the MapReduce job, the results reveal the average account balance associated with each specific client age. Below are the key observations:

* *Age-Wise Averages*: The output displays the average balance for each distinct age. For instance, clients aged 23 have an average balance of ₹2,117.95, while those aged 25 average around ₹1,240.05.
* *Emerging Trends:*
* Gradual Increase: A general upward trend can be observed, where average balances tend to rise with age—indicating growing financial stability over time.
* Notable Fluctuations: Some age groups deviate from this pattern, showing either unusually high or low balances, possibly reflecting lifestyle changes, income variability, or major life events.
* *Financial Diversity*: The data underscores substantial variability in average balances across age groups. This highlights the broad range of financial conditions and behaviours across different life stages.

*Conclusion:*

This age-wise balance analysis offers valuable insight into how financial standing evolves with age. By identifying both consistent patterns and anomalies, such analysis empowers financial institutions to design targeted services and personalized products that better align with the needs and financial habits of specific age demographics.

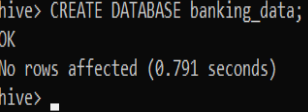
***Hive***

1. Data Ingestion and Table Creation:

* Create a Hive database named banking\_data.

Steps:

1. Start Hive CLI
2. Create a new database named banking\_data.



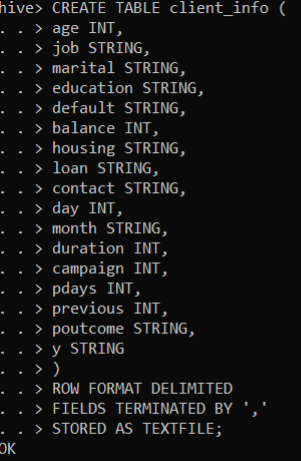
1. Switch to the newly created database:



* Define and create a Hive table client\_info with appropriate data types for the bank.csv dataset.

Steps:

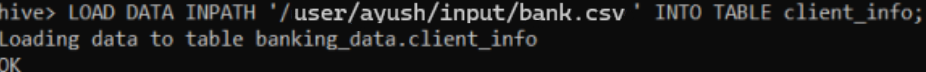
1. Create the client\_info table with appropriate data types based on the columns in the bank.csv file:



* Load the data from the bank.csv file into the client\_info table.

Steps:

1. Load the banking data from hadoop directory to the client\_info table.



1. Query the client\_info table to verify that the data has been loaded correctly.



Output:

A screenshot of a computer screen

AI-generated content may be incorrect.

1. Basic Data Exploration:

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

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.



Output:

A black background with white text

AI-generated content may be incorrect.

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.

* Display the first 10 rows of the dataset*.*

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.



Output:

A screenshot of a computer screen

AI-generated content may be incorrect.

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.

1. Data Filtering and Sorting:

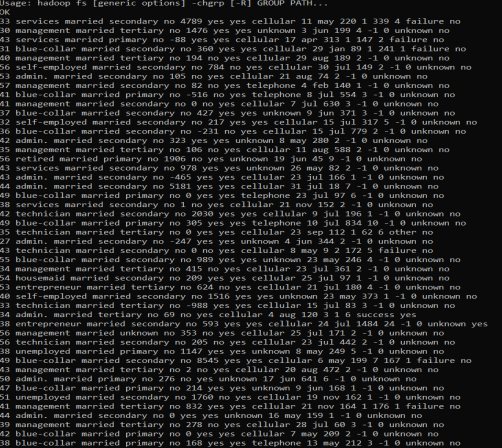
* Retrieve all records of clients who are married and have a personal loan.

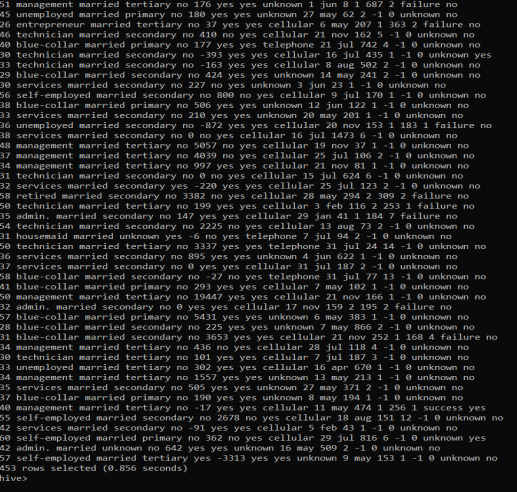
Steps:

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.



Output:





Summary of result:

This query filters the data to retrieve records of clients who are married and have a personal loan.

We can see the output is a list of all married clients who have taken a personal loan, including all columns of data.

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

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.



Output:

A screenshot of a computer screen

AI-generated content may be incorrect.

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.

1. Data Aggregation and Grouping:

* Calculate the average age of clients for each job category.

Steps:

1. 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.
2. 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.
3. 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



Output:

A computer screen shot of a number

AI-generated content may be incorrect.

Summary of the Results:

Here, we can see that the average age of most of the clients for different job categories is between 35 to 45.

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

Steps:

1. 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.
2. 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 defaulted.
3. GROUP BY education and default: The query then groups the filtered data by the education column and default column.
4. 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.
5. 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.



Output:

A black screen with white text

AI-generated content may be incorrect.

Summary:

Here, we can see that most of the clients who defaulted have a secondary education level.

1. Complex Queries for Insights:

* 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.

Steps:

1. Inner Query - Identifying Top 5 Job Categories by Average Balance
   * FROM client\_info: The query selects data from the client\_info table.
   * GROUP BY job: It groups the data by the job column, meaning it will aggregate data for each job category.
   * AVG (balance) AS avg\_balance: For each job category, the query calculates the average balance (AVG (balance)) and stores it as avg\_balance.
   * COUNT (\*) AS total\_clients: It counts the total number of clients in each job category using COUNT (\*) and stores this value as total\_clients.
   * 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.
   * 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.
   * LIMIT 5: The query limits the result to the top 5 job categories with the highest average balance.
2. Outer Query - Calculating Subscription Percentage

* FROM (...) sc: The outer query selects data from the result of the inner query, which is aliased as sc.
* SELECT sc.job, sc.avg\_balance: The outer query directly selects the job and avg\_balance columns from the inner query's results.
* (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.



Output:

A screenshot of a computer screen

AI-generated content may be incorrect.

Summary:

* 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.
* 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).

Steps:

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).
7. FROM (...) AS top\_month: The outer query selects data from the result of the inner query, which has been aliased as top\_month.
8. SELECT month, total\_contacts: The outer query selects the month and total\_contacts fields directly from the result of the inner query.
9. (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.



Output:

A black background with white text

AI-generated content may be incorrect.

Summary:   
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.

1. Correlation Analysis:

* Calculate the correlation between age and balance for the clients.

Steps:

1. The CORR (age, balance) function calculates the Pearson correlation coefficient between the age and balance columns.
2. This coefficient will range from -1 to 1, where:

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



Output:

A black background with white text

AI-generated content may be incorrect.

Summary:

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.

1. Trend Analysis:

* Analyze the year-over-year trend in the number of clients contacted.

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.

A black screen with white text

AI-generated content may be incorrect.

Output:

A screenshot of a computer

AI-generated content may be incorrect.

Summary:

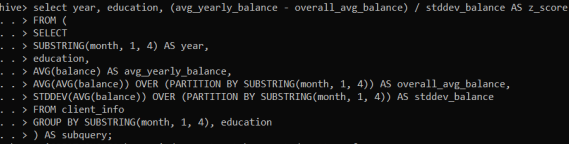
This output will give us a clear view of the year-over-year trend in the number of client contacts. Here, we did not have year information in the bank\_data.csv file. So, we could not get the desired result.

1. Anomaly Detection:

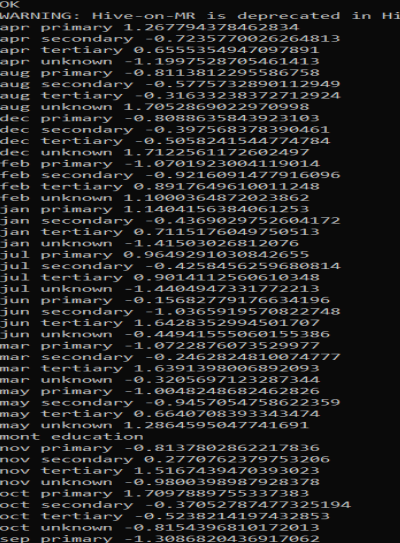
* Identify any unusual patterns in the average yearly balance across different education levels.

Steps:

1. 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).
2. education: Selects the education column from the client\_info table.
3. AVG (balance) AS avg\_yearly\_balance: Calculates the average balance for clients grouped by year and education.
4. 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.
5. 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.
6. GROUP BY SUBSTRING (month, 1, 4), education: Groups the data by year and education to calculate the metrics above.
7. (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.



Output:

 A computer screen shot of a number

AI-generated content may be incorrect.

Summary:

1. 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.
2. 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.
3. Advanced Analysis:

* 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.

Steps:

1. poutcome: This column indicates the outcome of the previous marketing campaign (e.g., 'success', 'failure', etc.).
2. count (\*) AS total\_clients: Counts the total number of clients for each poutcome category.
3. 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.
4. 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.
5. From: The query uses the client\_info table, which contains the relevant data.
6. Group By: Groups the results by the poutcome column, meaning that the calculations are done separately for each distinct value of poutcome.
7. Order By: Orders the results by subscription\_rate in descending order, showing the outcomes with the highest subscription rates first.

A screen shot of a computer

AI-generated content may be incorrect.

Output:

A screenshot of a computer screen

AI-generated content may be incorrect.

Summary:

The results of this query help to understand the effectiveness of previous marketing outcomes 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.

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

Steps:

1. 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).
2. AVG (duration) AS avg\_contact\_duration: Calculates the average duration of the contact in seconds for each subscription\_status.
3. From: The data is retrieved from the client\_info table.
4. Group By: The results are grouped by the subscription\_status.

A screen shot of a computer

AI-generated content may be incorrect.

Output:

A black background with white text

AI-generated content may be incorrect.

Summary:

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.