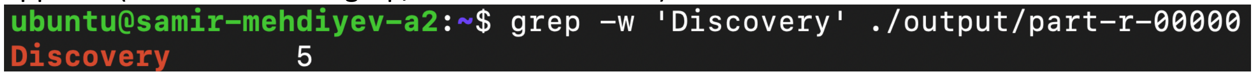
Task 1.1

a. Look at the contents of the folder "output", what files are placed in there? What do they mean?

In the output directory, Hadoop generates two files "_SUCCESS" and "part-r-00000". The "_SUCCESS" file is empty, and its presence indicates that the MapReduce job completed successfully. The "part-r-00000" file contains the actual output of your MapReduce job. In the case of the word count example, it should contain each unique word from the input files and its corresponding count. The naming convention "part-r-00000" is standard in Hadoop, where "part" signifies that it's a part of the output, "r" stands for reduce task (since the output is the result of a reduce task), and "00000" is just a partition number. In a larger job with multiple reducers, you might see multiple output files, like "part-r-00001", "part-r-00002", and so on.

b. Looking at the output files, how many times did the word 'Discovery' (case sensitive) appear? (hint: commands grep/cat could be useful)



```
ubuntu@samir-mehdiyev-a2:~$ grep -w 'Discovery' ./output/part-r-00000
Discovery       5
```

Based on the output files, the word 'Discovery' (case sensitive) appeared 5 times. I used grep command to search 'Discovery' string in the output file, and '-w' is used for searching only whole word, not as partition of some words.

c. In this example we used Hadoop in "Local (Standalone) Mode". What is the difference between this mode and the Pseudo-distributed mode?

Local (Standalone) Mode: In this mode, Hadoop runs on a single machine as a single Java process. None of the Hadoop daemons (such as NameNode, DataNode, etc.) run in this mode. This mode doesn't use HDFS (Hadoop Distributed File System), instead, it uses the local file system for all storage. It's primarily used for debugging, and it's the fastest among all modes. There's no need to configure any files (like hdfs-site.xml, core-site.xml) when running Hadoop in this mode.

Pseudo-distributed Mode: In this mode, Hadoop also runs on a single machine, but each Hadoop daemon runs in a separate Java process. This mode simulates a cluster on a small scale. It uses HDFS for managing input and output processes. Unlike the standalone mode, you need to change the configuration files (core-site.xml, hdfs-site.xml) for setting up the environment. This mode is used for both development and debugging.

In summary, the key difference between the two modes is that in Local Mode, all Hadoop processes run in a single JVM, making it suitable for debugging. In contrast, Pseudo-distributed Mode simulates a multi-node Hadoop cluster on a single machine, with each Hadoop daemon running in a separate JVM, making it suitable for both development and debugging.

Task 1.2

a. What are the roles of files core-site.xml and hdfs-site.xml?

core-site.xml: This file informs the Hadoop daemon where NameNode (the master node) runs in the cluster. It contains the configuration settings for Hadoop Core, such as I/O settings that are common to HDFS and MapReduce.

hdfs-site.xml: This file contains the configuration settings for HDFS daemons; the NameNode, and the DataNodes. Here, we can configure hdfs-site.xml to specify default block replication and permission checking on HDFS. The actual number of replications can also be specified when the file is created.

b. Describe the different services listed when executing 'jps'. What are their functions in Hadoop and HDFS?

NameNode: This is the master node that manages the file system namespace and regulates access to files by clients. It maintains the directory tree of all files in the file system, and tracks where across the cluster the file data is kept.

DataNode: These are the worker nodes that store and retrieve blocks when they are told to (by clients or the NameNode), and report back to the NameNode periodically with lists of blocks that they are storing.

Task 1.3

a. Explain the roles of the different classes in the file WordCount.java.
The `WordCount.java` file contains three classes: `TokenizerMapper`, `IntSumReducer`, and `WordCount`.

TokenizerMapper: This class extends `Mapper` and implements the `map` function. It takes a set of data and converts it into another set of data, where individual elements are broken down into tuples (key-value pairs). In the context of word count, it takes text input and breaks it down into words and emits a key-value pair for each word, where the key is the word, and the value is 1.

IntSumReducer: This class extends `Reducer` and implements the `reduce` function. It takes the output from the `map` function as input and combines those data tuples into a smaller set of tuples. In the context of word count, it sums up the values for each word key and emits a key-value pair with the word and its total count.

WordCount: This is the driver class that sets up and runs the job. It specifies the input and output paths, the mapper and reducer classes to use, and the types for the output key and value.

b. What is HDFS, and how is it different from the local file system on your VM?
HDFS, or Hadoop Distributed File System, is a distributed file system designed to run on commodity hardware. It is fault-tolerant, provides high throughput access to application data,

and is suitable for applications that have large data sets. HDFS operates by breaking down large data sets into smaller blocks, which are distributed across multiple nodes in the cluster.

The local file system (LFS), on the other hand, is a file system that manages the storage and retrieval of data on a single machine. Unlike HDFS, LFS does not distribute data across multiple nodes and is not designed to handle large data sets.

Task 1.5

1. One example of JSON formatted data is Twitter tweets: https://dev.twitter.com/overview/api/tweets. Based on the twitter documentation, how would you classify the JSON-formatted tweets - structured, semi-structured or unstructured data?

JSON-formatted tweets from Twitter's API can be classified as semi-structured data. This is because they exhibit characteristics of both structured and unstructured data. On one hand, these tweets have a consistent set of fields like user ID, tweet text, timestamp, and other metadata, which are indicative of structured data. However, the JSON format allows for a more flexible schema compared to traditional relational databases. This flexibility means that some tweets might include additional fields (e.g., location data, hashtags) that others don't, and the data can also be hierarchical with nested elements, such as embedded retweet details or user information. This mix of standardized and variable elements, along with the integration of different data types (text, numbers, booleans), places JSON-formatted tweets firmly in the semi-structured category.

2. Elaborate on pros and cons for SQL and NoSQL solutions, respectively. Give some examples of particular data sets/scenarios that might be suitable for these types of databases. (expected answer length: 0.5 A4 pages)
When considering SQL and NoSQL databases, their respective pros and cons align with different types of data scenarios and use cases.

SQL databases, like MySQL, Oracle, or Microsoft SQL Server, are structured and use a schema to define data formats, making them ideal for scenarios where data integrity and relationships between the data are crucial. Their structured query language enables complex queries with precision, which is essential in environments like financial systems where transactions must be accurately tracked and reported. SQL databases ensure ACID compliance (Atomicity, Consistency, Isolation, Durability), vital for maintaining data integrity in critical applications. For example, in a banking system, SQL databases are used for maintaining customer accounts, transactions, and balances, where the relationships between these entities are complex and data accuracy is paramount. Another example is healthcare systems, where patient records, treatment histories, and appointment schedules are intricately related and require consistent and reliable data handling.

However, SQL databases can struggle with scalability, especially with very large volumes of data or horizontal scaling. Their rigid schema makes it difficult to adapt to evolving data needs without significant redesign or downtime.

NoSQL databases, such as MongoDB, Cassandra, or Couchbase, offer greater flexibility and are well-suited for handling large volumes of unstructured or semi-structured data. They excel in scenarios requiring rapid scalability and can efficiently manage diverse data types. For instance, social media platforms like Twitter or Instagram deal with various types of data (text, images, videos), and the data structure can change rapidly. NoSQL databases can store and manage this heterogeneous data efficiently, and scale to handle the high volume of data generated by millions of users. In the realm of big data and real-time analytics, where data formats can be varied and change quickly, NoSQL databases provide the necessary speed and flexibility. For example, in IoT applications, where devices produce vast amounts of diverse data, NoSQL databases can handle the scalability and variety effectively.

However, NoSQL databases might lack the maturity and robust feature set of SQL databases, and their flexible schema can sometimes lead to data inconsistency issues. They may also require more complex queries or a deeper understanding for optimal use.

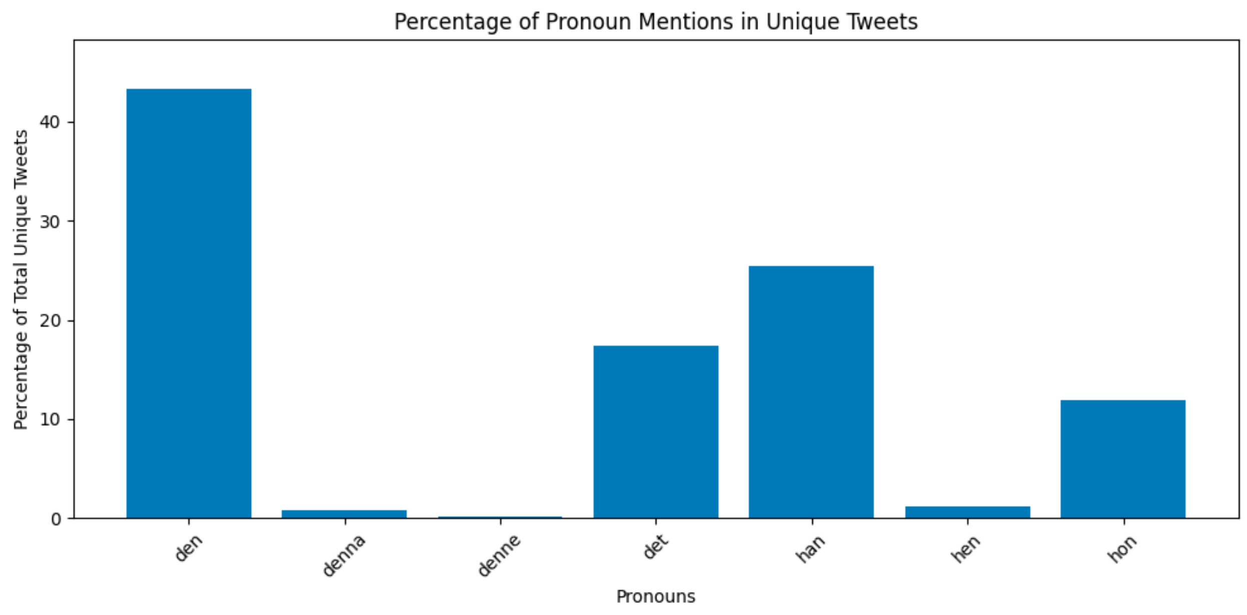
Task 2.1: Hadoop MapReduce Approach

We employed Hadoop Streaming to analyze approximately 5 million JSON-formatted tweets, specifically targeting occurrences of Swedish pronouns. The `mapper.py` script was utilized to process each tweet, filter out retweets, and identify pronouns. The `reducer.py` script then aggregated these occurrences. Here is the steps for execution:

1. Upload the dataset to HDFS.
   ```
   hadoop-3.3.6/bin/hdfs dfs -put /home/ubuntu/input /user/ubuntu
   ```
2. Ensure `mapper.py` and `reducer.py` are executable.
   ```
   chmod +x mapper.py reducer.py
   ```
3. Run the MapReduce job using Hadoop's streaming JAR, specifying the mapper and reducer scripts.
   ```
   hadoop-3.3.6/bin/hadoop jar hadoop-3.3.6/share/hadoop/tools/lib/hadoop-
   streaming-3.3.6.jar \
   -files /home/ubuntu/mapper.py,/home/ubuntu/reducer.py \
   -mapper ‘python3 mapper.py’ \
   -reducer ‘python3 reducer.py’ \
   -input /user/ubuntu/input \
   -output /user/ubuntu/tweets_output
   ```

4. Transferred the output from HDFS to the local file system.
   ```
   hadoop-3.3.6/bin/hdfs dfs -copyToLocal /user/ubuntu/tweets_output
   /home/ubuntu/local_output
   ```

5. Visualized the results using a bar chart in Python, highlighting that 'den' was the most frequently used pronoun.



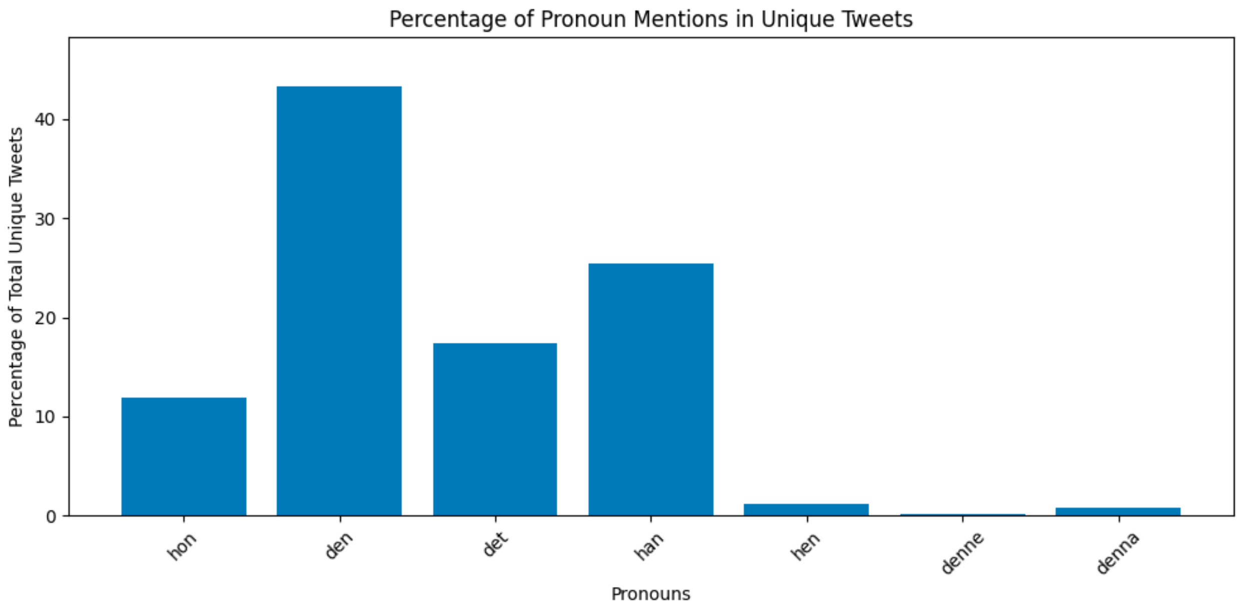
Task 2.2: MongoDB Approach

We replicated the analysis using MongoDB, a NoSQL database. The steps involved:

1. Install MongoDB on our system.
   sudo apt-get install gnupg curl
   **curl -fsSL https://www.mongodb.org/static/pgp/server-7.0.asc | **
   sudo gpg --dearmor -o /usr/share/keyrings/mongodb-server-7.0.gpg
   echo "deb [arch=amd64,arm64 signed-by=/usr/share/keyrings/mongodb-server-7.0.gpg] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/7.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-7.0.list
   sudo apt-get update
   sudo apt-get install -y mongodb-org
2. Start the MongoDB service and ensure it is active.
   sudo systemctl start mongodb
   sudo systemctl enable mongodb
3. Import the tweet data into a MongoDB database using a looped `mongoimport` command. For loop just used for importing all files in one command.
   ```
   for file in /home/ubuntu/input/*.txt; do
     mongoimport --db my_db --collection tweets --file "$file"
   done
   ```
4. Install `pymongo` to interface with MongoDB from Python.
   pip3 install pymongo
5. Created a script, `task2_2.py`, which connected to the database, filtered tweets, and counted pronoun occurrences with a similar logic to the Hadoop approach.

python3 task2_2.py

6. Extracted the results to a file for visualization. Using 'task2_1.ipynb' visualize the results.



In both tasks I got the same results, and the final visualization demonstrated that 'den' was the predominant pronoun in the dataset.

What are some pros and cons of the MongoDB solution compared to the implementation you did in Task 2.1?

While the Hadoop approach is robust for batch-processing large datasets, it requires setting up a distributed computing environment and is more complex. MongoDB offers a simpler setup and real-time processing capabilities, making it a more user-friendly option for real-time applications and smaller datasets. However, for massive datasets where in-memory processing is not feasible, Hadoop's distributed computing prowess offers a scalable solution.