task 1.1. ex. 4. Answer the following questions:

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

- reference: a stackoverflow post

-
  ```
  output/
  ├── _SUCCESS
  └── part-r-00000
  ```
- The `_SUCCESS` file is created in the output directory when the job has completed without errors.
- The `part-r-00000` file contains the output (stdout) of the job.

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

-
  ```
  $ grep Discovery output/part-r-00000
  Discovery 5
  ```

  i.e. Discovery appeared 5 times.

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

- reference: a geeksforgeeks post
- In local/standalone mode, Hadoop runs on a single machine without the distributed architecture, without the use of HDFS.
- In pseudo-distributed mode, Hadoop also runs on a single machine, but with the distributed architecture, utilizing HDFS.

task 1.2. ex. 4. Answer the following questions:

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

- core-site.xml: Contains common hadoop settings, like information on how to connect to the hdfs.
- hdfs-site.xml: Contains hdfs-specific settings, e.g. in this exercise the number of replicates for each block of data is set to 1, so the data is only stored in one location.

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

- reference: HDFS documentation, Oracle jps documentation
- listed are:

  ```
  $ jps
  19143 Jps
  19067 NameNode
  18827 DataNode
  ```

  - jps:

    ```
    $ man -k jps
    jps (1)              - list the instrumented JVMs on the target system
    ```

    lists all java processes running on the machine, so this arguably does not have much to do with hadoop at all, but because hadoop runs in the JVM this command can show which components of hadoop are running on the system.

  - NameNode: HDFS master server, manages the whole filesystem through block operations and metadata management. Some filesystem operations (e.g. file opening, renaming) are handled by this master.

  - DataNode: Servers storing the actual data of the HDFS. These servers handle the data store and retrieve operations.

task 1.3 ex. 6. Answer the following questions:

code referenced in task description is here

a. Explain the roles of the different classes in the file WordCount.java.

- WordCount: root/main class, contains entry point (WordCount.main) which sets up the job pipeline (configures which code/class to use for map/combine/reduce/output, also sets input and output path for job) and executes the job (starts it, then waits for its completion)
- TokenizerMapper: split input text into individual words (and creates an associated count value initialized to 1)
- IntSumReducer: takes stream of input values with same key (key here is word from mapper) and sums their associated count values, then outputs the word and the sum of the count values for this word

b. What is HDFS, and how is it different from the local file system on your VM?

- a local file system makes hardware from the same machine available
- HDFS is a distributed file system, which makes hardware from other machines available
  - it is scalable across a large number of devices and the storage is pooled into a common namespace, effectively providing a single large storage device
  - it is also designed for fault-tolerance, large files and streaming data access

task 1.4

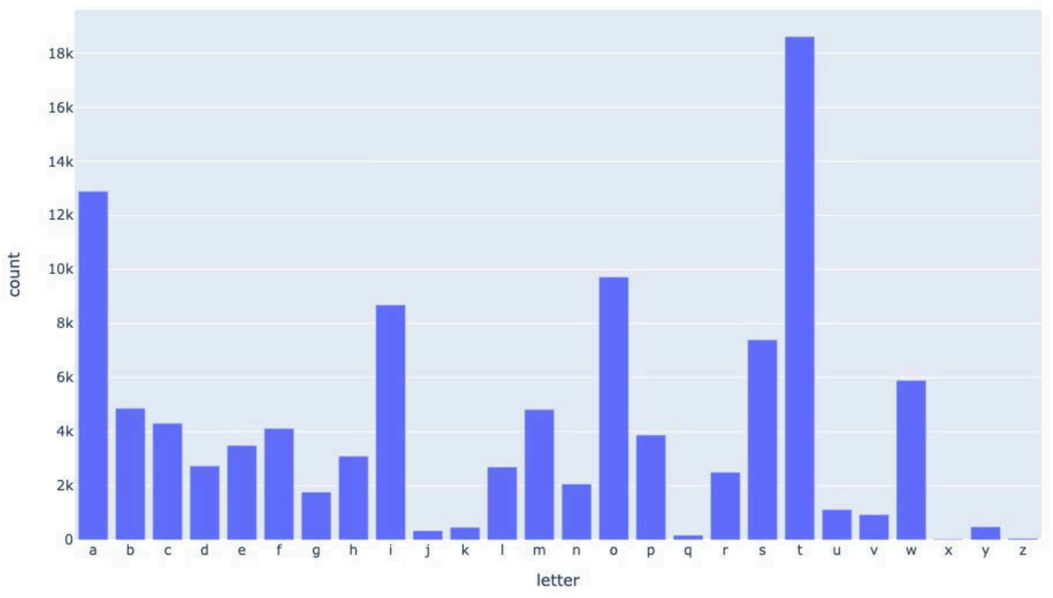
1. Modify the above example code in WordCount.java so that it, based on the same input files, counts the occurrences of words starting with the same first letter (ignoring case). The output of the job should thus be a file containing lines like (if there were 1234 words beginning with A or a):

   ```
   while (itr.hasMoreTokens()) {
     word.set(itr.nextToken());
     context.write(word, one);
   }
   ```

   - to (my solution):

     ```
     while (itr.hasMoreTokens()) {
       String nextword=itr.nextToken();
       char firstChar=Character.toLowerCase(nextword.charAt(0));
       String firstCharAsString=String.valueOf(firstChar);
       word.set(firstCharAsString);
       context.write(word, one);
     }
     ```
2. Make a plot that shows the counts for each letter (i.e. excluding all non-letter symbols) and include that in your report. Do not include the entire output file.
   - get result to make plot:

     ```
     $ hadoop-3.3.6/bin/hdfs dfs -cat pseudoDistributedWordCountT14FixedJar/part-
     r-00000 | grep "^[a-z]"
     a 12893
     < snip >
     z 55
     ```



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?
    - Semi-structured. The few definitions I found for structured data require some tabular layout (e.g. Forbes), which does not apply here. All tweets are in the JSON format though, itself a well-defined data format, and the tweet objects also have many fields in common, e.g. id and text, so there is some structure.
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)

Using SQL for data storage and query has the advantage that, as the name implies, it handles structured data. Structured data can be stored much more efficiently than unstructured data, so it may take up less storage space than the same information would if it was in an unstructured format. This can also make querying this data much more memory and compute efficient, so query results may be returned faster than they might be from unstructured data. The structure of an SQL database also makes it very suitable for large scale distributed storage. The downside of SQL is that not all data fits well into the data types SQL provides, even extending to the structure of whole SQL databases, e.g very heterogenous data may not be well suitable, and so while it could be stored in an SQL database, storing and querying might be less performant than in other non-SQL formats.

NoSQL can have the same advantages and disadvantages, but also any other property, depending on the implementation. There may be a NoSQL implementation highly specific to a single dataset, so storage and querying might be more performant here than if this data was stored in an SQL database, but then the downside is that this implementation is specific to a single dataset, requiring higher maintance. Another implementation may be less specific, be less performant than SQL, but very fast to implement for a new dataset.

SQL is a good fit for scenarios where large amounts of homogeneous data are stored long-term, changed infrequently at most, and consist of a small fraction of numerical data and short text fragments (i.e. no dataframe-scale numerical data, rather individual numbers, and small text fragment rather than large text chunks), like business records, transaction logs, or other archival data.

NoSQL is a good fit for semi-structured or unstructured data that consists of neither single numbers nor short text fragments, like large bodies of text (e.g. whole books, webpages), images or audio. It is also a good fit for very heterogeneous data, like a product catalogue not specific to any category (where most products have no common properties).

task 2.1

mapper:

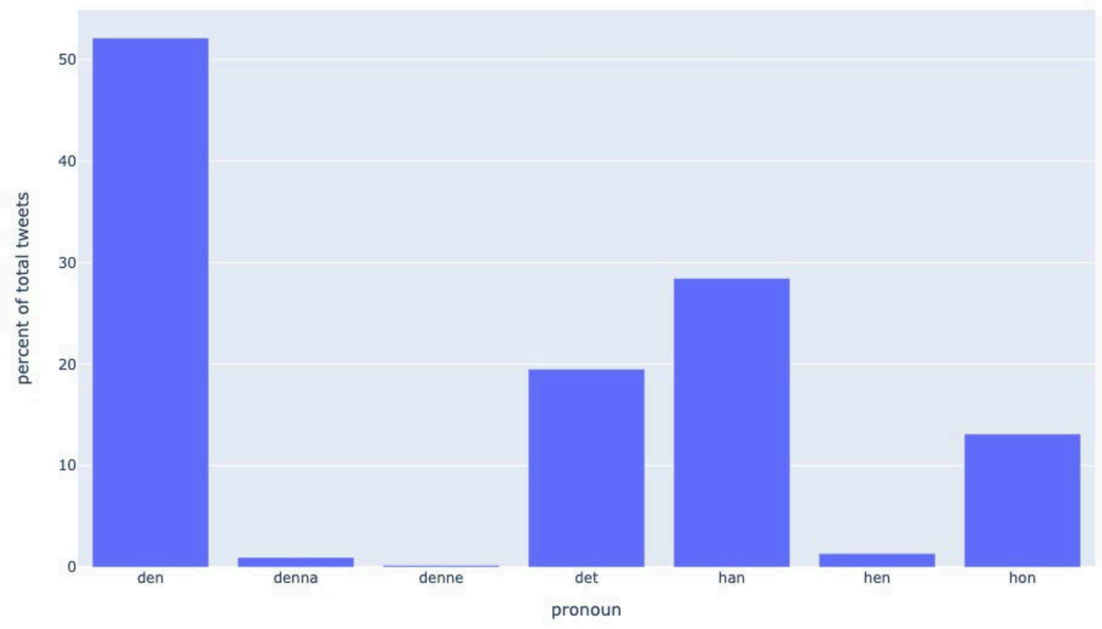
- discards empty input lines, parses remaining as json objects
- retweets are discarded (identified by the presence of the 'retweeted_status' json field)
- tweet text is searched for all occurences of any pronoun (using a lengthy regular expression), pronoun duplicates are removed (upper/lowercase spelling is counted as the same pronoun)
- for each tweet, a 'pronoun' (not actually a pronoun, rather the string/key: "tweets") is emitted to count the total number of tweets using the map/reduce approach
- notes on libraries used: orjson is used to parse the json objects, faster than cpython's json library. the regex library is used instead of cpython re, because re does not support the unicode character groups `\p{}`.
- note on the choice of the particular regular expressions: this expression is compatible with MongoDB, and yields the exact same matches there (a simpler option using cpython's re library is to use word boundaries `\b`, though those are not compatible with MongoDB).

reducer:

- splits each input line into a pronoun and a number (number of occurences of this pronoun in this instance)
- an internal dictionary is used to keep track of the cumulative sum of all occurences of each pronoun
- when stdin is empty, the entries of the dictionary are written to stdout (a pronoun, its total number of occurences per output line, and the fraction of all tweets containing this pronoun)

result:

```
den 1220703 ( 52.13% of total)
denna 22475 (  0.96% of total)
denne 3988  (  0.17% of total)
det 456539  ( 19.50% of total)
han 666483  ( 28.46% of total)
hen 31150 (  1.33% of total)
hon 307084  ( 13.11% of total)
tweets  2341577 (100.00% of total)
```



task 2.2

My MongoDB implementation is fundamentally very similar to the implementation using Hadoop in task 2.1. The main differences are that the input files are not read from stdin, but instead read within the python script itself, and that the regular expression to find the pronouns in each tweet are not matched in cpython, rather each tweet is ingested into the MongoDB database, where all tweets are queried at once.

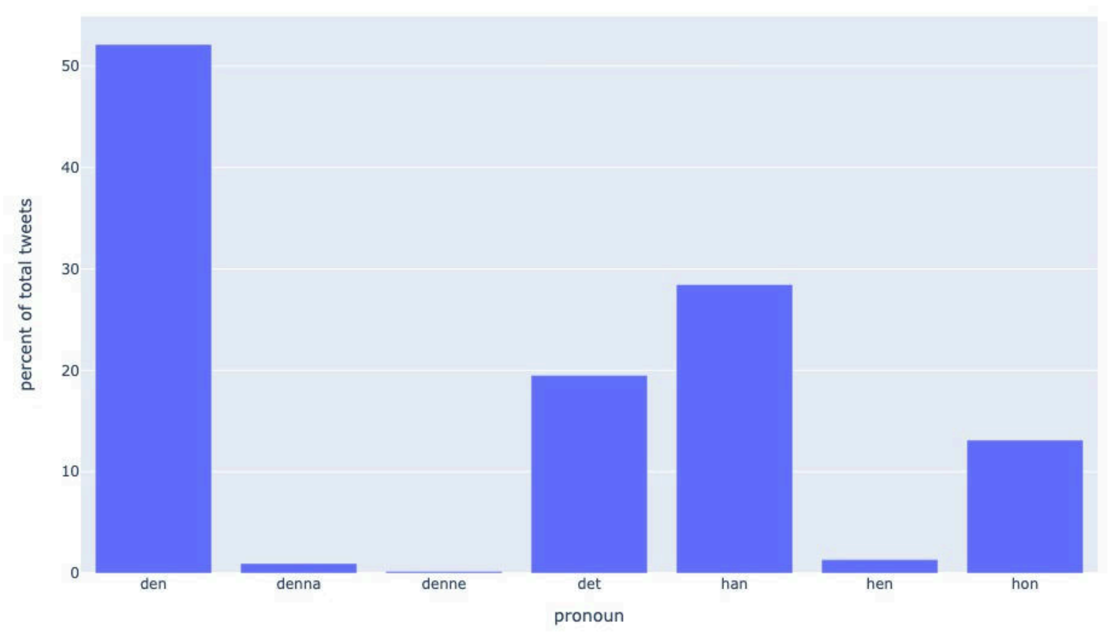
The main advantage of using MongoDB here is that a large fraction of the total compute in this exercise is used to parse of the tweet json objects. The parsed json objects are stored in MongoDB. This allows executing multiple queries on all tweets without having to parse all tweets repeatedly. This can be quite useful e.g. when debugging the regular expression used to match the pronouns in each tweet because the results do not match the cpython re results.

The main disadvantage of MongoDB here is that MongoDB uses a custom query structure. While this query structure is surely optimized for high performance on its internal data representation, these queries are much less flexible than python and any external library that could be used to run queries in task 2.1. For example, in python there are many libraries that offer regular expression execution, with support for different regex components, while MongoDB offers a single regex implementation, which lacks support for some features (like word boundaries).

results:

```
total number of tweets: 2341577
den 1220703 (52.13% of total)
denna 22475 (0.96% of total)
denne 3988 (0.17% of total)
det 456539 (19.50% of total)
han 666483 (28.46% of total)
hen 31150 (1.33% of total)
hon 307084 (13.11% of total)
```

These are the same results as in task 2.1, so this plot also looks the same.



percent of total tweets
0
10
20
30
40
50
den
denna
denne
det
han
hen
hon
pronoun