

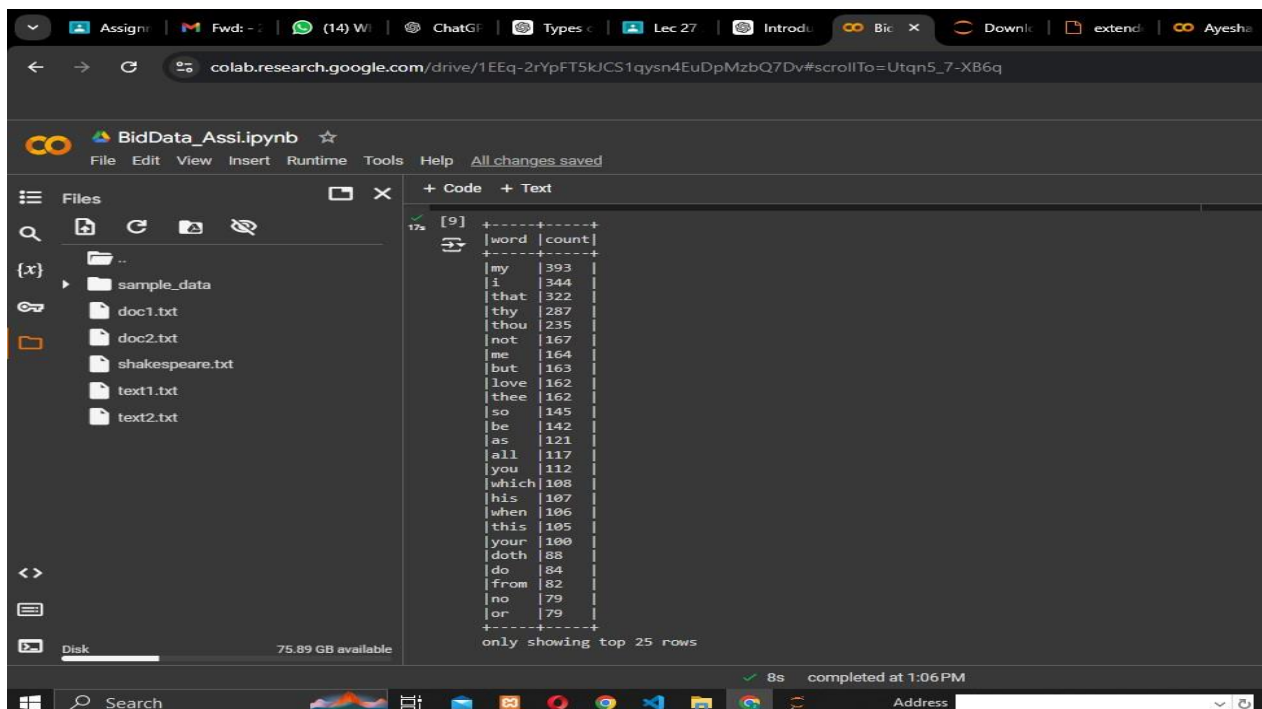
Word Count in Spark

Objective:

This mini project focuses on applying Spark to perform word count operations and analyze the results on a document

1. Single File Analysis:

25 most common words from the single file



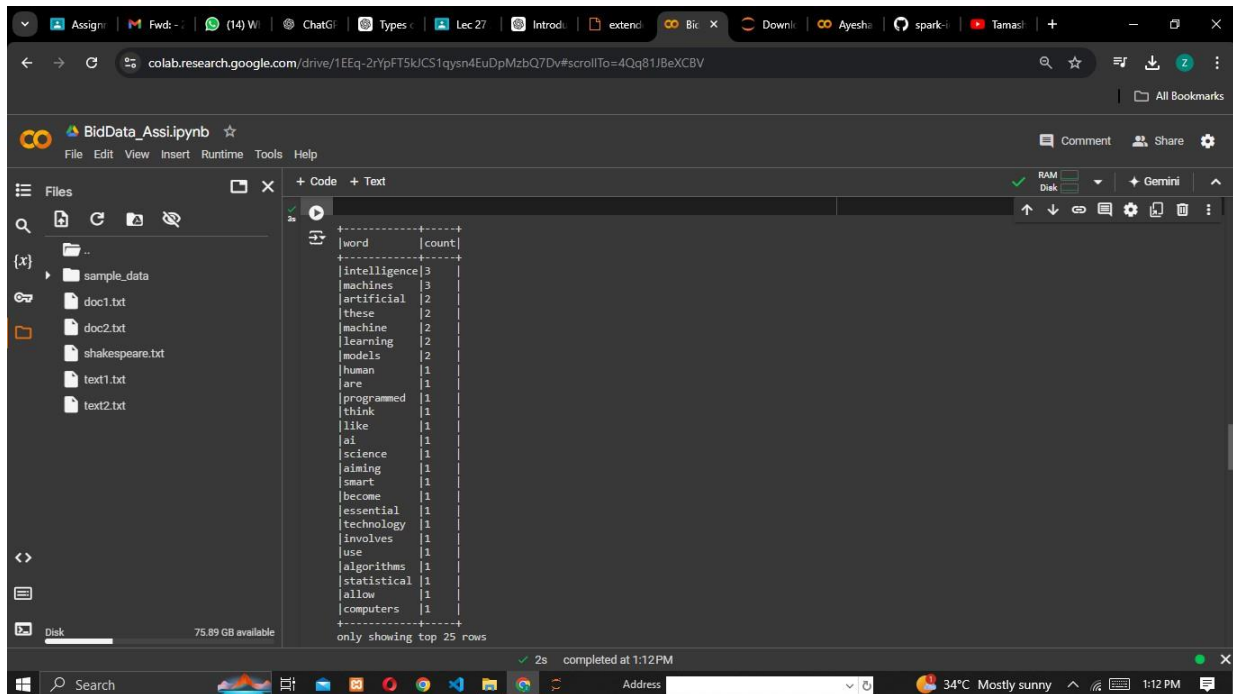
The screenshot shows a Google Colab notebook titled "BidData_Assi.ipynb". The left sidebar displays a file explorer with a folder named "sample_data" containing several text files: "doc1.txt", "doc2.txt", "shakespeare.txt", "text1.txt", and "text2.txt". The main area of the notebook shows a code cell that has been executed, resulting in a table of word counts. The table has two columns: "word" and "count". The words are listed in descending order of frequency, with "my" having the highest count at 393. The table shows the top 25 words, and a message at the bottom indicates "only showing top 25 rows". The status bar at the bottom of the notebook shows "8s" and "completed at 1:06 PM".

word	count
my	393
i	344
that	322
thy	287
thou	235
not	167
me	164
but	163
love	162
thee	162
so	145
be	142
as	121
all	117
you	112
which	108
his	107
when	106
this	105
your	100
doth	88
do	84
from	82
no	79
or	79

2. Multiple Files Analysis:

25 most common words from the multiple files.

Extended Word Count:

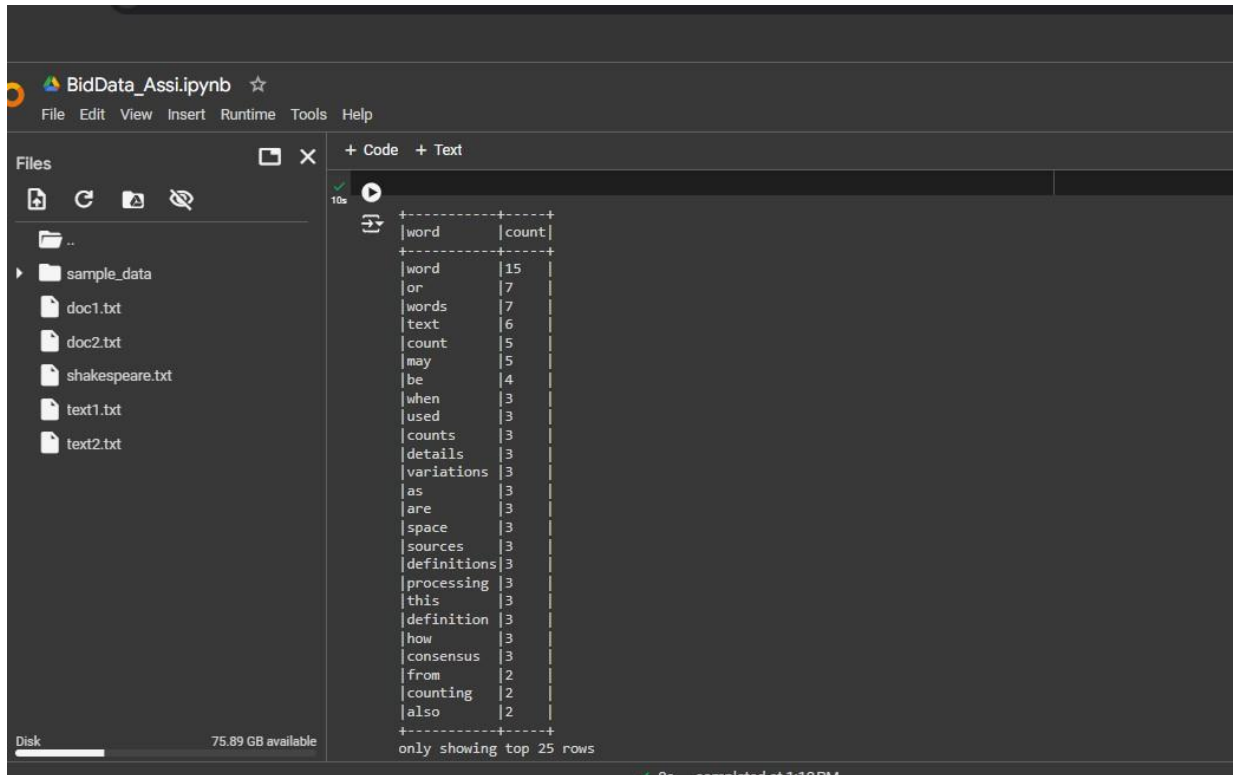


The screenshot shows a Google Colab notebook titled "BidData_Ass1.ipynb". The left sidebar displays a file explorer with a folder named "sample_data" containing several text files: "doc1.txt", "doc2.txt", "shakespeare.txt", "text1.txt", and "text2.txt". The main code cell contains a table of the 25 most common words and their counts, sorted in descending order. The table is as follows:

word	count
intelligence	3
machines	3
artificial	2
these	2
machine	2
learning	2
models	2
human	1
are	1
programmed	1
think	1
like	1
ai	1
science	1
aiming	1
smart	1
become	1
essential	1
technology	1
involves	1
use	1
algorithms	1
statistical	1
allow	1
computers	1

Below the table, it states "only showing top 25 rows". The bottom status bar indicates the code was completed in 2 seconds at 1:12 PM. The system tray at the very bottom shows the temperature as 34°C and the weather as "Mostly sunny".

Word Count on a Document:



The screenshot shows a Jupyter Notebook titled 'BidData_Assi.ipynb'. The left sidebar displays a file explorer with a folder 'sample_data' containing five text files: 'doc1.txt', 'doc2.txt', 'shakespeare.txt', 'text1.txt', and 'text2.txt'. The main area shows a code cell with a word count program. The output of the program is a table with two columns: 'word' and 'count'. The table lists 25 words and their counts, sorted by count in descending order. The words and their counts are: word (15), or (7), words (7), text (6), count (5), may (5), be (4), when (3), used (3), counts (3), details (3), variations (3), as (3), are (3), space (3), sources (3), definitions (3), processing (3), this (3), definition (3), how (3), consensus (3), from (2), counting (2), and also (2). The output is truncated, showing only the top 25 rows.

```
word count
word 15
or 7
words 7
text 6
count 5
may 5
be 4
when 3
used 3
counts 3
details 3
variations 3
as 3
are 3
space 3
sources 3
definitions 3
processing 3
this 3
definition 3
how 3
consensus 3
from 2
counting 2
also 2
```

only showing top 25 rows

3. Handling Faults in Spark:

How Spark handles node failures during the execution of your word count program.

Spark handles node failures by using lineage information to recompute lost data and reschedule tasks on available nodes. This fault tolerance ensures that the system can recover and continue processing without major interruptions.

Advantages did you observe using Spark compared to traditional methods like MapReduce?

Spark offers several advantages over traditional MapReduce, including faster data processing due to in-memory computation, more advanced data handling with support for complex operations, and a more user-friendly API for iterative tasks and interactive queries. This leads to improved performance and ease of use in handling large-scale data processing tasks.

4. Performance Questions:

Performance Optimizations and Impact:

1. In-Memory Computation:

Leveraging Spark's in-memory processing with DataFrames instead of RDDs can significantly speed up the computation, especially for iterative tasks.

2. Efficient File Handling:

By processing multiple files in a single RDD operation, the program reduces overhead and improves performance compared to processing files individually.

3. Optimized DataFrame Operations:

Converting RDDs to DataFrames for final sorting and displaying leverages Spark SQL's optimizations, which can enhance query performance and execution time.

Impact:

These optimizations lead to faster execution times and better resource utilization, especially when handling large datasets or multiple files simultaneously.