
CSE587: Data Intensive Computing

Project - Phase 3 - Report

Name	UB-ID	UB Person No.	UB Email
Rachna Bhatia	rachnatr	50486853	rachnatr@buffalo.edu
Rounak Biswas	rounakbi	50450467	rounakbi@buffalo.edu
Shruti Gupta	sgupta55	50485160	sgupta55@buffalo.edu
Sujan Reddy	sujanred	50471029	sujanred@buffalo.edu

Using HADOOP

Loading the data in Hadoop

For the Hadoop setup, we created a local or remote instance of Hadoop successfully, providing a distributed computing environment for big data processing.

To confirm the successful setup of the Hadoop instance, we executed the '*jps*' command which displayed the running components, including the NameNode, DataNode, ResourceManager, NodeManager, and Jps.

```
C:\Hadoop\sbin>start-dfs.cmd

C:\Hadoop\sbin>jps
17072 Jps
5540 NameNode
20104 DataNode

C:\Hadoop\sbin>start-yarn.cmd
starting yarn daemons

C:\Hadoop\sbin>jps
5540 NameNode
14840 NodeManager
18872 Jps
20104 DataNode
16476 ResourceManager
```




By executing the command '*hdfs dfs -mkdir /DIC_Project*', we created a directory named DIC_Project successfully in the Hadoop Distributed File System (HDFS).

```
C:\Hadoop\sbin>hdfs dfs -mkdir /DIC_Project
```





localhost:9870/explorer.html#/

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities ▾

Browse Directory

/ Go!   

Show 25 entries Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	drwxr-xr-x	bhati	supergroup	0 B	Jul 07 00:59	0	0 B	DIC_Project	
<input type="checkbox"/>	drwxr-xr-x	bhati	supergroup	0 B	Jul 07 01:04	0	0 B	output	
<input type="checkbox"/>	drwx-----	bhati	supergroup	0 B	Jul 07 01:02	0	0 B	tmp	

Showing 1 to 3 of 3 entries Previous 1 Next

Hadoop, 2022.

We used the command `'hdfs dfs -ls /'` to confirm the presence of the created directories. It displayed three items, including directories named DIC_Project, output, and tmp, along with their respective permissions, ownership, and creation timestamps in HDFS.

```
C:\Hadoop\sbin>hdfs dfs -ls /
Found 3 items
drwxr-xr-x - bhati supergroup      0 2023-07-07 00:59 /DIC_Project
drwxr-xr-x - bhati supergroup      0 2023-07-07 01:04 /output
drwx----- - bhati supergroup      0 2023-07-07 01:02 /tmp
```

We loaded the data file "netflix_titles.csv" successfully into Hadoop using the command `'hdfs dfs -put source_path destination_path'`.

```
C:\Hadoop\sbin>hdfs dfs -put C:\Users\bhati\OneDrive\Documents\DocumentsOfRachna\DIC\ProjectNotebooks\netflix_titles.csv /DIC_Project
```

We verified the successful loading of the file by accessing the browser interface, which confirmed the presence of the loaded data in the target destination directory, "DIC_Project".

Browse Directory

/DIC_Project

Show 25 entries

Search:

Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name
-rw-r--r--	bhati	supergroup	2.3 MB	Jul 07 00:59	3	128 MB	netflix_titles.csv

Showing 1 to 1 of 1 entries

Previous 1 Next

Hadoop, 2022.

We also tried executing the command `'hdfs dfs -rm -r /output/wordcount_result'` to check if the file named "wordcount_result" located at the path "/output" in HDFS is getting deleted successfully.

```
C:\Hadoop\sbin>hdfs dfs -rm -r /output/wordcount_result
Deleted /output/wordcount_result
```

We measured the time taken for loading the data using the command `'echo %TIME% && command && cmd /v:on /c echo !TIME!'` which gave the start time and end time for the task.

```
C:\Users\bhati\OneDrive\Documents\DocumentsOfRachna>echo %TIME% && hdfs dfs -put C:\Users\bhati\OneDrive\Documents\DocumentsOfRachna\ProjectNotebooks\netflix_titles.csv /DIC_Project && cmd /v:on /c echo !TIME!
4:38:50.59
4:38:52.91
```

Time taken to load the data = End time - Start time = 0:0:2.32

Loading the data using Pandas Dataframe

We loaded the data using pandas' `pd.read_csv()` function and calculated the time taken for loading the data using the `time.time()` function.

```
start_time = time.time()

test_df_load = pd.read_csv('netflix_titles.csv')

end_time = time.time()
time_difference_pandas = end_time - start_time
print("Loading the data on the pandas dataframe - ", end_time - start_time)
```

Loading the data on the pandas dataframe - 0.07274699211120605

Time taken to load the data on hadoop - 2.32

Time taken to load the data using pandas dataframe - 0.07

Time difference between both = 2.32 - 0.07 = 2.25

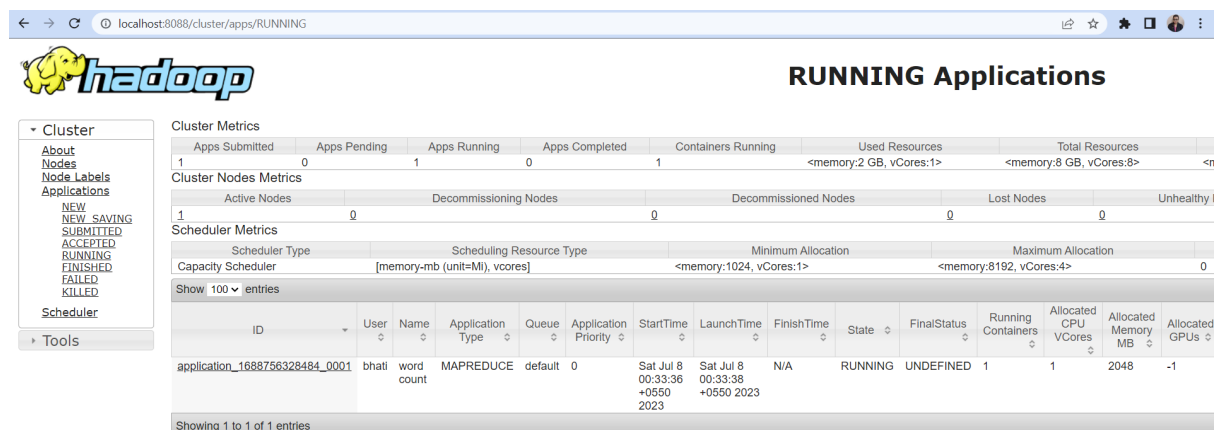
Word Count Using MapReduce

We applied a mapping dictionary `replace_map` to the 'country' column of the dataframe, replacing specific country names with their corresponding standardised values. We then extracted the modified 'country' column, replaced commas with spaces, and filled any missing values with 'NoData' to prepare the data for the word count analysis using MapReduce.

We executed the command `'hadoop jar C:\Hadoop\share\hadoop\mapreduce\hadoop-mapreduce-examples-3.2.4.jar wordcount /DIC_Project/country.csv /output/wordcount_result'` to perform the word count operation using the MapReduce jar file.

```
C:\Hadoop\sbin>hadoop jar C:\Hadoop\share\hadoop\mapreduce\hadoop-mapreduce-examples-3.2.4.jar wordcount /DIC_Project/country.csv /output/wcForCountry
2023-07-08 00:33:35,087 INFO client.RMProxy: Connecting to ResourceManager at /0.0.0.0:8032
2023-07-08 00:33:35,581 INFO mapreduce.JobResourceUploader: Disabling Erasure Coding for path: /tmp/hadoop-yarn/staging/bhati/.staging/job_1688756328484_0001
2023-07-08 00:33:35,760 INFO input.FileInputFormat: Total input files to process : 1
2023-07-08 00:33:35,824 INFO mapreduce.JobSubmitter: number of splits:1
2023-07-08 00:33:36,342 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1688756328484_0001
2023-07-08 00:33:36,346 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-07-08 00:33:36,543 INFO conf.Configuration: resource-types.xml not found
2023-07-08 00:33:36,544 INFO resource.ResourceUtils: Unable to find 'resource-types.xml'.
2023-07-08 00:33:36,955 INFO impl.YarnClientImpl: Submitted application application_1688756328484_0001
2023-07-08 00:33:37,010 INFO mapreduce.Job: The url to track the job: http://Himesh:8088/proxy/application_1688756328484_0001/
2023-07-08 00:33:37,012 INFO mapreduce.Job: Running job: job_1688756328484_0001
2023-07-08 00:33:45,256 INFO mapreduce.Job: Job job_1688756328484_0001 running in uber mode : false
2023-07-08 00:33:45,258 INFO mapreduce.Job: map 0% reduce 0%
2023-07-08 00:33:50,398 INFO mapreduce.Job: map 100% reduce 0%
2023-07-08 00:33:55,468 INFO mapreduce.Job: map 100% reduce 100%
2023-07-08 00:33:56,510 INFO mapreduce.Job: Job job_1688756328484_0001 completed successfully
2023-07-08 00:33:56,640 INFO mapreduce.Job: Counters: 54
```

We verified in the browser interface that the job is running on the Hadoop cluster, leveraging the distributed processing capabilities of Hadoop.



The screenshot displays the Hadoop cluster management interface in a web browser. The browser address bar shows `localhost:8088/cluster/apps/RUNNING`. The interface features the Hadoop logo and a sidebar with navigation options: Cluster, About, Nodes, Node Labels, Applications, NEW, NEW SAVING, SUBMITTED, ACCEPTED, RUNNING, FINISHED, FAILED, KILLED, Scheduler, and Tools. The main content area is titled "RUNNING Applications" and contains several sections: Cluster Metrics, Cluster Nodes Metrics, Scheduler Metrics, and a table of running applications. The Cluster Metrics section shows 1 app submitted, 0 pending, 1 running, 0 completed, and 1 container running. The Cluster Nodes Metrics section shows 1 active node, 0 decommissioning nodes, 0 decommissioned nodes, 0 lost nodes, and 0 unhealthy nodes. The Scheduler Metrics section shows the scheduler type as Capacity Scheduler and the scheduling resource type as memory-mb (unit=Mi, vcores). The table of running applications shows one entry: application_1688756328484_0001, user bhati, name word count, application type MAPREDUCE, queue default, application priority 0, start time Sat Jul 8 00:33:36 +0550 2023, launch time Sat Jul 8 00:33:38 +0550 2023, finish time N/A, state RUNNING, final status UNDEFINED, running containers 1, allocated CPU vCores 1, allocated memory MB 2048, and allocated GPUs -1.

ID	User	Name	Application Type	Queue	Application Priority	StartTime	LaunchTime	FinishTime	State	FinalStatus	Running Containers	Allocated CPU vCores	Allocated Memory MB	Allocated GPUs
application_1688756328484_0001	bhati	word count	MAPREDUCE	default	0	Sat Jul 8 00:33:36 +0550 2023	Sat Jul 8 00:33:38 +0550 2023	N/A	RUNNING	UNDEFINED	1	1	2048	-1

We confirmed the successful completion of the wordcount job using MapReduce by checking the final status as "succeeded" on the Hadoop cluster's browser interface. The job took approximately **18 seconds** to complete, as indicated by the elapsed time recorded during the execution.

Application Overview	
User:	bhati
Name:	word count
Application Type:	MAPREDUCE
Application Tags:	
Application Priority:	0 (Higher Integer value indicates higher priority)
YarnApplicationState:	FINISHED
Queue:	default
FinalStatus Reported by AM:	SUCCEEDED
Started:	Sat Jul 08 06:40:09 +0530 2023
Launched:	Sat Jul 08 06:40:10 +0530 2023
Finished:	Sat Jul 08 06:40:27 +0530 2023
Elapsed:	18sec
Tracking URL:	History
Log Aggregation Status:	DISABLED
Application Timeout (Remaining Time):	Unlimited
Diagnostics:	
Unmanaged Application:	false
Application Node Label expression:	<Not set>
AM container Node Label expression:	<DEFAULT_PARTITION>

The output file containing the word counts for countries was stored at the path `/output/wordcount_result` as specified.

localhost:9870/explorer.html#/output

Hadoop Overview Datanodes Datanode Volume Failures Snapshot Startup Progress Utilities

Browse Directory

/output

Go!

Show 25 entries

Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	drwxr-xr-x	bhati	supergroup	0 B	Jul 07 01:05	0	0 B	wordcount_result	

Showing 1 to 1 of 1 entries

Previous 1 Next

Hadoop, 2022.

We've attached the output file '*wordcount_result*' as well.

Word Count Using Pandas

We performed the word count using the pandas dataframe on the 'country' column by modifying it since you wanted to clean the data a little.

Word count using Pandas for country column

```
[7] start_time_wordcount_panda = time.time()

combined_text = ' '.join(test_df_load['country'].astype(str).tolist())
combined_text = combined_text.replace(","," ")
combined_text = combined_text.replace("United States", "UnitedStates")
combined_text = combined_text.replace("United Kingdom", "UnitedKingdom")
combined_text = combined_text.replace("United Arab Emirates", "UnitedArabEmirates")
combined_text = combined_text.replace("Hong Kong", "HongKong")
combined_text = combined_text.replace("New Zealand", "NewZealand")
combined_text = combined_text.replace("Saudi Arabia", "SaudiArabia")
combined_text = combined_text.replace("Czech Republic", "CzechRepublic")
combined_text = combined_text.replace("South Korea", "SouthKorea")
combined_text = combined_text.replace("West Germany", "WestGermany")
combined_text = combined_text.replace("East Germany", "EastGermany")
combined_text = combined_text.replace("Cayman Islands", "CaymanIslands")
combined_text = combined_text.replace("South Africa", "SouthAfrica")
combined_text = combined_text.replace("Soviet Union", "SovietUnion")
combined_text = combined_text.replace("Sri Lanka", "SriLanka")
combined_text = combined_text.replace("Vatican City", "VaticanCity")
combined_text = combined_text.replace("Dominican Republic", "DominicanRepublic")

word_counts = pd.Series(combined_text.split()).value_counts()

end_time_wordcount_panda = time.time()

time_difference_wordcount_panda = end_time_wordcount_panda - start_time_wordcount_panda
```

The word counts for countries were as follows -

```
[8] word_counts

UnitedStates    3690
India           1046
nan              831
UnitedKingdom    806
Canada           445
...
Bermuda          1
Ecuador          1
Armenia          1
Mongolia         1
Montenegro       1
Length: 125, dtype: int64
```

The time taken by the pandas is shown below -

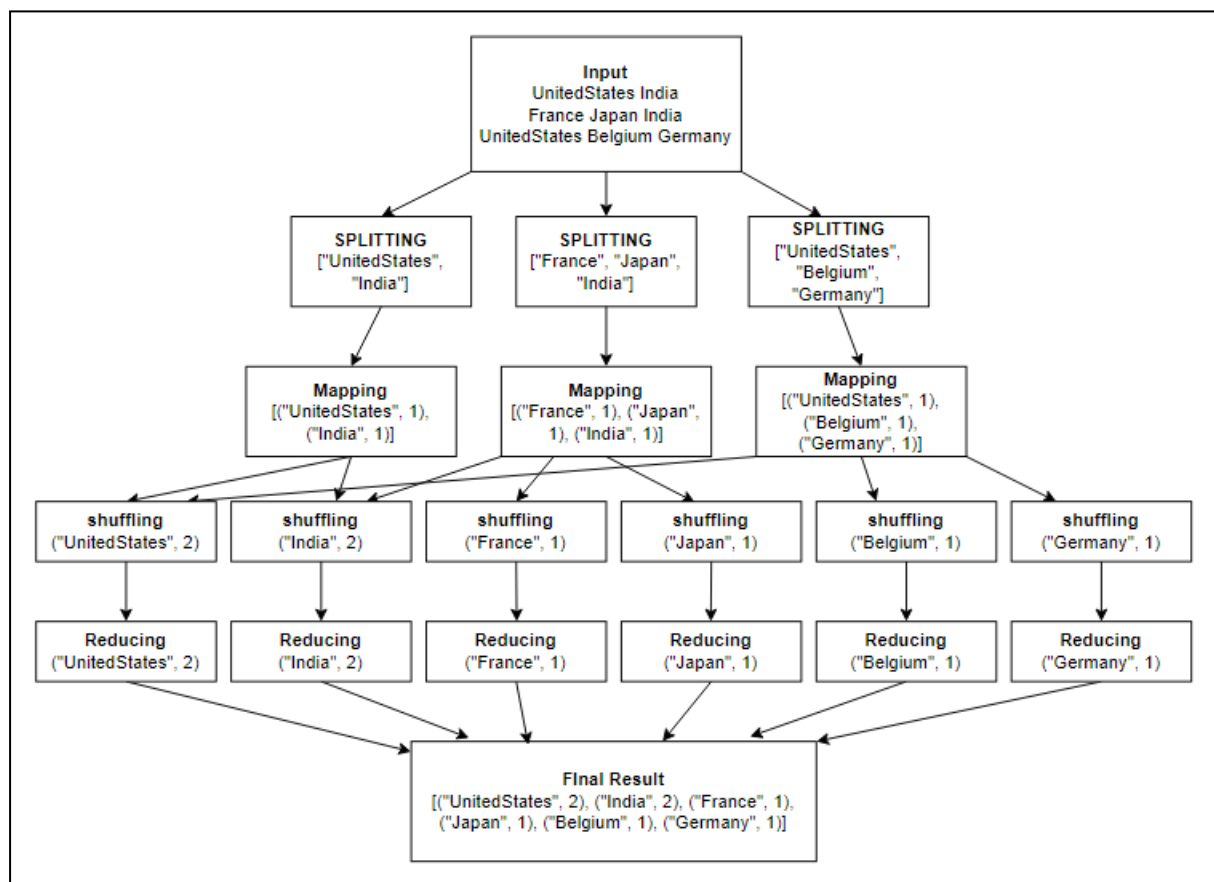
```
print("Time for data loading using pandas: ", time_difference_pandas)
print("Time for data loading using pyspark: ", time_difference_pyspark)
print("Time difference between pyspark and pandas dataframe: ", time_difference_pyspark - time_difference_pandas)

Time for data loading using pandas: 0.13768243789672852
```

Working of MapReduce

Here we have taken the columns of the countries as the input. According to the size of the input file, the input data is often split or broken into smaller parts. Each split is a logical chunk of the data that may be handled separately.

We have given the flow chart diagram below for the random 3 of the rows on the columns of the countries - (UnitedStates, India), (France, Japan, India) and (UnitedStates, Belgium, Germany)



FLOWCHART

Splitting:

First, all 3 rows are splitted during the splitting phase in parallel.

Mapping:

Each split is handled in parallel by many map jobs during the mapping phase.

Each map job reads the split that has been given to it and tokenizes it into separate words.

The map task emits a key-value pair for each word it encounters, with the key being the word itself and the value often being a constant like [("UnitedStates", 1), ("India", 1)]

Sorting and Shuffling:

The intermediate key-value pairs are the result of the map tasks.

In a shuffling and sorting phase, the MapReduce gathers all intermediate values related to the same key. For instance, ("India", 2).

The key-value pairs are sorted and then shuffled for the reduced jobs.

Reducing:

A different key and its corresponding values are processed by each reduction job.

For the word count example, the reduce task repeatedly iterates over and sums the data (often a list) associated with each word.

The word serves as the key and its associated count serves as the value in the final output that the reduction job outputs.

Output:

The results of the word count are often sent to a file (wordcount_result) as the reduction jobs' final output.

Using SparkContext

Loading data using pyspark vs pandas.

First, we loaded our dataset with more than 6000 rows to the pyspark dataframe and recorded the time taken to load the data.

Loading with PySpark

```
[3] start_time_pyspark = time.time()

spark = SparkSession.builder.getOrCreate()

df = spark.read.csv("netflix_titles.csv", header=True)

end_time_pyspark = time.time()

time_difference_pyspark = end_time_pyspark - start_time_pyspark

pandas_df = df.toPandas()

spark.stop()
```

This is the output of the spark dataframe converted to pandas dataframe.

pandas_df.head()

	show_id	type	title	director	cast	country	date_added	release_year	rating	duration	listed_in	description
0	s1	Movie	Dick Johnson Is Dead	Kirsten Johnson	None	United States	September 25, 2021	2020	PG-13	90 min	Documentaries	As her father nears the end of his life, filmm...
1	s2	TV Show	Blood & Water	None	Ama Qamata, Khosi Ngema, Gali Mablane, Thaban...	South Africa	September 24, 2021	2021	TV-MA	2 Seasons	International TV Shows, TV Dramas, TV Mysteries	After crossing paths at a party, a Cape Town t...
2	s3	TV Show	Ganglands	Julien Leclercq	Sami Bouajila, Tracy Gotoas, Samuel Jouy, Nabil...	None	September 24, 2021	2021	TV-MA	1 Season	Crime TV Shows, International TV Shows, TV Act...	To protect his family from a powerful drug lor...
3	s4	TV Show	Jailbirds New Orleans	None	None	None	September 24, 2021	2021	TV-MA	1 Season	Docuseries, Reality TV	Feuds, flirtations and toilet talk go down amo...
4	s5	TV Show	Kota Factory	None	Mayur More, Jitendra Kumar, Ranjan Raj, Alam K...	India	September 24, 2021	2021	TV-MA	2 Seasons	International TV Shows, Romantic TV Shows, TV ...	In a city of coaching centers known to train l...

Then, we loaded the same dataset to pandas dataframe and recorded the time taken to load the data.

Loading data using Pandas

```
[5] start_time = time.time()
test_df_load = pd.read_csv('netflix_titles.csv')
end_time = time.time()
time_difference_pandas = end_time - start_time
```

Both these times were printed and the difference to load the data was recorded as below:

Time difference of loading our dataset with PySpark and Pandas

```
[13] print("Time for data loading using pandas: ", time_difference_pandas)
      print("Time for data loading using pyspark: ", time_difference_pyspark)
      print("Time difference between pyspark and pandas dataframe: ", time_difference_pyspark - time_difference_pandas)

Time for data loading using pandas: 0.13768243789672852
Time for data loading using pyspark: 29.427265644073486
Time difference between pyspark and pandas dataframe: 29.289583206176758
```

Word count using sparkcontext vs pandas on 'country' column

Using SparkContext

We needed to clean the data a little. So, we removed the commas and corrected the name of the countries to make them appear without a space. And then performed map reduce (using SparkContext) on the cleaned column.

Word count using SparkContext for country column

```
[9] scnew = SparkContext("local", "WordCount")

country_rdd = scnew.parallelize(pandas_df['country'].astype(str).tolist())

start_time_wordcount_sparkc = time.time()
cleaned_rdd = country_rdd.map(lambda line: line.replace(", ", " ")
                             .replace("United States", "UnitedStates")
                             .replace("United Kingdom", "UnitedKingdom")
                             .replace("United Arab Emirates", "UnitedArabEmirates")
                             .replace("Hong Kong", "HongKong")
                             .replace("New Zealand", "NewZealand")
                             .replace("Saudi Arabia", "SaudiArabia")
                             .replace("Czech Republic", "CzechRepublic")
                             .replace("South Korea", "SouthKorea")
                             .replace("West Germany", "WestGermany")
                             .replace("East Germany", "EastGermany")
                             .replace("Cayman Islands", "CaymanIslands")
                             .replace("South Africa", "SouthAfrica")
                             .replace("Soviet Union", "SovietUnion")
                             .replace("Sri Lanka", "SriLanka")
                             .replace("Vatican City", "VaticanCity")
                             .replace("Dominican Republic", "DominicanRepublic"))
```

For mapping, each line of the text is taken and split into separate words. Then, it assigns a unique number (here, 1) to each word.

For reducing, words that are the same are grouped together and their occurrence is counted. This step combined the counts for each word from different parts of the column.

So, using these map and reduce operations, we performed word count on the country column. And also noted the time taken for the process.

```

word_counts = cleaned_rdd.flatMap(lambda line: line.split()) \
    .map(lambda word: (word, 1)) \
    .reduceByKey(lambda a, b: a + b) \
    .sortBy(lambda x: x[1], ascending=False)

word_counts = word_counts.collect()
end_time_wordcount_sparkc = time.time()

scnew.stop()

time_difference_wordcount_sparkc = end_time_wordcount_sparkc - start_time_wordcount_sparkc

```

The result i.e., word_counts using SparkContext was also printed -

```

[10] word_counts

[('UnitedStates', 3690),
 ('India', 1046),
 ('nan', 831),
 ('UnitedKingdom', 806),
 ('Canada', 445),
 ('France', 393),
 ('Japan', 318),
 ('Spain', 232),
 ('SouthKorea', 231),
 ('Germany', 226),
 ('Mexico', 169),
 ('China', 162),
 ('Australia', 160),
 ('Egypt', 117),
 ('Turkey', 113),
 ('HongKong', 105),
 ('Nigeria', 103),
 ('Italy', 100),
 ('Brazil', 97),
 ('Argentina', 91),
 ('Belgium', 90),
 ('Indonesia', 90),
 ('Taiwan', 89),
 ('Philippines', 83),
 ('Thailand', 70),
 ('SouthAfrica', 62),
 ('Colombia', 52),
 ('Netherlands', 50),

```

Using Pandas

We needed to clean the data a little. So, we removed the commas and corrected the name of the countries to make them appear without a space. And then word count (by splitting text using space) was performed on the cleaned country column to obtain the count of each country appearing in the column. Time taken for this operation was also noted.

Word count using Pandas for country column

```
[7] start_time_wordcount_panda = time.time()

combined_text = ' '.join(test_df_load['country'].astype(str).tolist())
combined_text = combined_text.replace(","," ")
combined_text = combined_text.replace("United States", "UnitedStates")
combined_text = combined_text.replace("United Kingdom", "UnitedKingdom")
combined_text = combined_text.replace("United Arab Emirates", "UnitedArabEmirates")
combined_text = combined_text.replace("Hong Kong", "HongKong")
combined_text = combined_text.replace("New Zealand", "NewZealand")
combined_text = combined_text.replace("Saudi Arabia", "SaudiArabia")
combined_text = combined_text.replace("Czech Republic", "CzechRepublic")
combined_text = combined_text.replace("South Korea", "SouthKorea")
combined_text = combined_text.replace("West Germany", "WestGermany")
combined_text = combined_text.replace("East Germany", "EastGermany")
combined_text = combined_text.replace("Cayman Islands", "CaymanIslands")
combined_text = combined_text.replace("South Africa", "SouthAfrica")
combined_text = combined_text.replace("Soviet Union", "SovietUnion")
combined_text = combined_text.replace("Sri Lanka", "SriLanka")
combined_text = combined_text.replace("Vatican City", "VaticanCity")
combined_text = combined_text.replace("Dominican Republic", "DominicanRepublic")

word_counts = pd.Series(combined_text.split()).value_counts()

end_time_wordcount_panda = time.time()

time_difference_wordcount_panda = end_time_wordcount_panda - start_time_wordcount_panda
```

Output for word count using pandas -

```
[8] word_counts

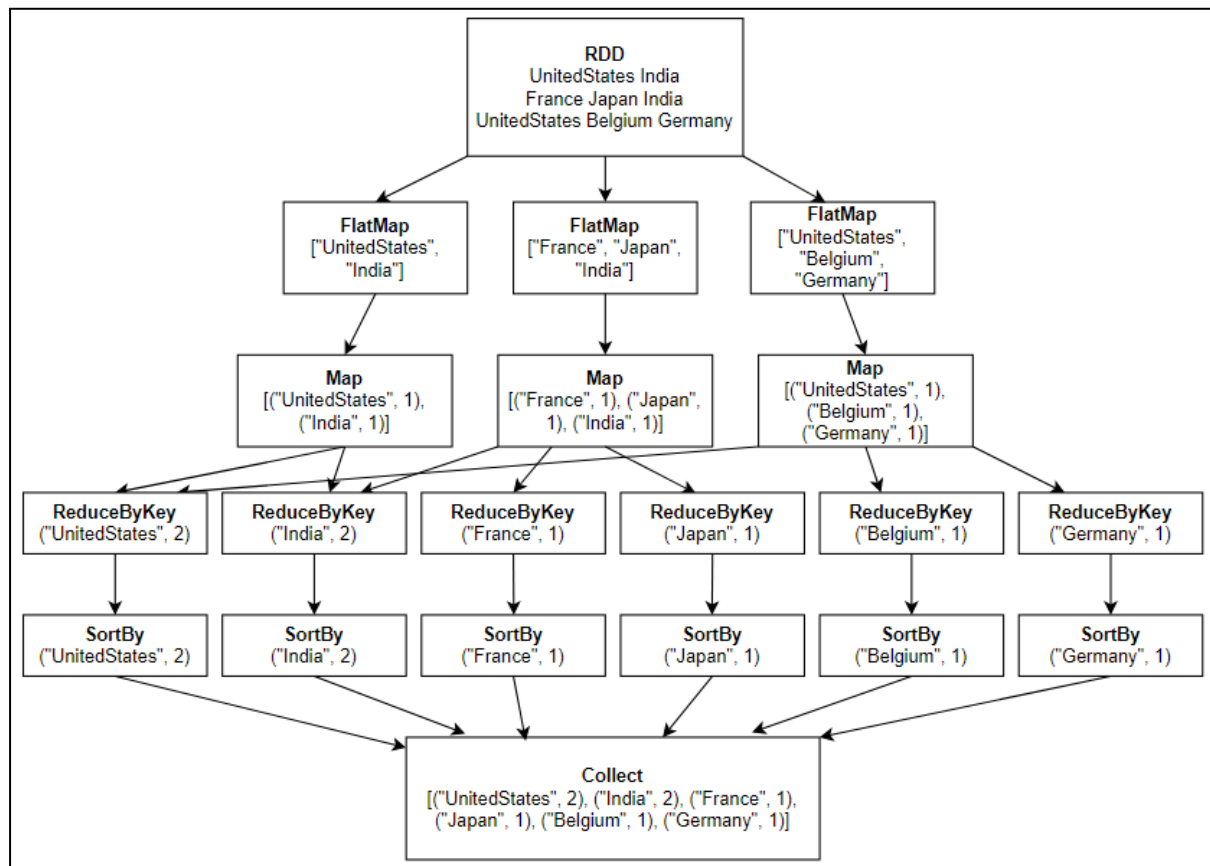
UnitedStates    3690
India           1046
nan              831
UnitedKingdom   806
Canada          445
...
Bermuda          1
Ecuador          1
Armenia          1
Mongolia         1
Montenegro       1
Length: 125, dtype: int64
```

Time difference for word count using Pandas vs SparkContext for country column

```
[11] print("Time for word count for country using pandas: ", time_difference_wordcount_panda)
print("Time for word count country using SparkContext: ", time_difference_wordcount_sparkc)
print("Time difference between pyspark and pandas dataframe: ", time_difference_wordcount_sparkc - time_difference_wordcount_panda)

Time for word count for country using pandas:  0.011105775833129883
Time for word count country using SparkContext:  3.609203577041626
Time difference between pyspark and pandas dataframe:  3.598097801208496
```

Flowchart for MapReduce using SparkContext



Explanation of the flowchart:

- RDD represents the initial RDD (Resilient Distributed Dataset) obtained from the 'country' column for our dataset. It contains the country values from that column.
- After this, the FlatMap operation is applied to the above RDD. It splits each line in the 'country' column, into individual words (countries).
- The next operation is the Map operation is performed on the output of the above step. It transforms each word (country) into a key-value pair. The country becomes the key, and an initial count of 1 is assigned as the value for each key-value pair.
- Following this, the ReduceByKey operation is applied to the output of the above step. It groups the key-value pairs based on the country and applies a reducing function that aggregates the counts for each country. This means that after this step, we have counts for each unique country.
- Now, the SortBy operation is performed on the key-value pairs obtained from the previous step. It sorts the key-value pairs in descending order based on the count. This sorting ensures that the countries are arranged based on their count, from highest to lowest.
- Finally, the Collect action retrieves the sorted key-value pairs and returns them as a local list. And hence we obtain a final list of key-value pairs, listing the word count.

Using Spark

In this section we chose the description column, and performed word count on that column with and without spark. We have loaded our dataset into a data frame called as 'netflix_df'. and converted the data frame into a spark data frame using spark variable.

WordCount for the column of description - Top 20

```
spark = SparkSession.builder.appName("WordCount").getOrCreate()

netflix_df = spark.read.csv("netflix_titles.csv", header=True, inferSchema=True)

words_df = netflix_df.select("description").withColumn("words", explode(split(col("description"), " ")))
word_count_df = words_df.groupBy("words").count().orderBy("count", ascending=False)

word_count_df.show()

spark.stop()
```

words	count
a	6891
the	5494
to	4412
and	4372
of	3826
in	2617
his	2307
with	1454
her	1383
an	1168
on	1161
for	1153
their	1116
A	1011
from	807
is	786
by	710
as	679
this	664
who	621

For "discription" - all word counts

```
spark = SparkSession.builder.appName("NetflixWordCount").getOrCreate()
netflix_df = spark.read.csv("netflix_titles.csv", header=True, inferSchema=True)

words_df = netflix_df.select("description").withColumn("words", explode(split(lower(trim(col("description"))), " ")))

word_count_df = words_df.groupBy("words").count().orderBy("count", ascending=False) #Getting the word counts
word_count_df.show(word_count_df.count(), truncate=False) # printing all word count results

spark.stop()
```

words	count
a	7902
the	5760
to	4475
and	4379
of	3826
in	3050
his	2313
with	1529
her	1388
an	1350
on	1228
for	1164
their	1122
when	1004
this	966
from	854

Here we have taken the word count for the column description of the dataset NETFLIX_TITLES.CSV and taken the variable words_df to select the description column and converted to lowercase and later used the group by and count function to count the total count for each of the words.

Code for word count without using PySpark

```
netflix_df = pd.read_csv("netflix_titles.csv")

# Converting the combined text into lowercase and then split the data on the whitespaces
combined_text = ' '.join(netflix_df['description']).lower()
words = combined_text.split()

word_counts = Counter(words) # Counting the number of words
word_counts_ordered = dict(sorted(word_counts.items(), key=lambda x: x[1], reverse=True)) # Descending order

for word, count in word_counts_ordered.items(): # Print the word count results in Descending Order
    print(f"{word}: {count}")

a: 7967
the: 5838
to: 4507
and: 4449
of: 3860
in: 3112
his: 2339
with: 1546
her: 1394
an: 1365
on: 1238
for: 1176
their: 1129
when: 1017
this: 1002
from: 869
is: 799
as: 779
by: 719
```

Here for this to find out the word count without using the pyspark we have taken the description column and first converted it into the lower case and later used the split function on the data and used the counter functionality to count the word_counts and used the dict to make it in the reverse order and printed the word count of each of the words in the reverse order.

Time comparison of word count with and without using PySpark

Execution time without PySpark – 0.030176877975463867
Execution time with PySpark – 0.010632991790771484

Here after executing using the pandas we have compared the results of the time execution without pyspark and with using the pyspark and we can see the results in the above picture.

Extra Credit

1. Loading data and Performing model training and Sentiment Analysis

Loading the data and taking first 2000 rows of the description column

```
[26] netflix_data = pd.read_csv('netflix_titles.csv')
      df = netflix_data['description']
```

Cleaning and preprocessing the data

Converted text to lowercase, special characters and punctuation were removed and stop words were removed

```
stopwords_list = set(stopwords.words('english'))
def textcleaning(text):
    text = text.lower()
    text = re.sub(r'^a-zA-Z\s', '', text)
    text = ' '.join(word for word in text.split() if word not in stopwords_list)
    return text

clean_df = df.apply(textcleaning)
```

Training Word2Vec model

```
[28] sentences = [description.split() for description in df]
      model = Word2Vec(sentences, window=5, min_count=1, workers=4)
      model.save("word2vec_model.bin")
```

Performing Sentiment Analysis on cleaned Description column

```
[29] model = Word2Vec.load("word2vec_model.bin")
      sia = SentimentIntensityAnalyzer()
      sentiments = []
      for description in clean_df:
          vector_sum = sum(model.wv.get_vector(word) for word in description.split() if word in model.wv.key_to_index)
          sentiment = sia.polarity_scores(' '.join(vector_sum.astype(str)))
          sentiments.append(sentiment)

      positive_count = sum(1 for sentiment in sentiments if sentiment['compound'] > 0)
      negative_count = sum(1 for sentiment in sentiments if sentiment['compound'] < 0)
      neutral_count = len(sentiments) - positive_count - negative_count
```

```
print("Sentiment Analysis Results:")
print("Positive: ", positive_count)
print("Negative: ", negative_count)
print("Neutral: ", neutral_count)
```

```
➤ Sentiment Analysis Results:
Positive: 965
Negative: 862
Neutral: 173
```

Printing the first 6 Descriptions and their Predicted Sentiment

```
for i in range(6):
    print(i + 1, "Description:", df[i])
    print("Predicted Sentiment:", sentiments[i])
    print()
```

1 Description: father nears end life filmmaker kirsten johnson stages death inventive comical ways help face inevitable
Predicted Sentiment: {'neg': 0.199, 'neu': 0.663, 'pos': 0.138, 'compound': -0.296}

2 Description: crossing paths party cape town teen sets prove whether privateschool swimming star sister abducted birth
Predicted Sentiment: {'neg': 0.174, 'neu': 0.684, 'pos': 0.142, 'compound': -0.1531}

3 Description: protect family powerful drug lord skilled thief mehdi expert team robbers pulled violent deadly turf war
Predicted Sentiment: {'neg': 0.406, 'neu': 0.399, 'pos': 0.196, 'compound': -0.7783}

4 Description: feuds flirtations toilet talk go among incarcerated women orleans justice center new orleans gritty reality series
Predicted Sentiment: {'neg': 0.176, 'neu': 0.653, 'pos': 0.171, 'compound': 0.2263}

5 Description: city coaching centers known train indias finest collegiate minds earnest unexceptional student friends navigate campus life
Predicted Sentiment: {'neg': 0.0, 'neu': 0.686, 'pos': 0.314, 'compound': 0.7506}

6 Description: arrival charismatic young priest brings glorious miracles ominous mysteries renewed religious fervor dying town desperate believe
Predicted Sentiment: {'neg': 0.215, 'neu': 0.594, 'pos': 0.192, 'compound': 0.128}

2. Performing Similarity Search

For the Word - "Love"

```
search_query = 'love'
if search_query in model.wv.key_to_index:
    similar_words = model.wv.most_similar(search_query, topn = 10)
    print(f"\nWords similar to '{search_query}' - ")
    for word, similarity in similar_words:
        print(f"{word} - {similarity}")
else:
    print(f"\n'{search_query}' is not in the vocabulary.")
```

Words similar to 'love' -

new	-	0.7501369118690491
back	-	0.7387745380401611
finds	-	0.7321494221687317
must	-	0.724092960357666
man	-	0.7176949381828308
find	-	0.7097305655479431
woman	-	0.7017538547515869
one	-	0.7011669278144836
family	-	0.6952736973762512
life	-	0.6774609684944153

For the Word - "Animated"

```
|: search_query = 'animated'
if search_query in model.wv.key_to_index:
    similar_words = model.wv.most_similar(search_query, topn = 10)
    print(f"\nWords similar to '{search_query}' - ")
    for word, similarity in similar_words:
        print(f"{word} - {similarity}")
else:
    print(f"\n'{search_query}' is not in the vocabulary.")
```

```
Words similar to 'animated' -
kancharapalem - 0.4066334366798401
beautiful - 0.3458538353443146
batch - 0.34569427371025085
posh - 0.34472644329071045
account - 0.3439401388168335
ku - 0.33144423365592957
bridgerton - 0.3265025317668915
known - 0.32336777448654175
reawakens - 0.32222798466682434
12 - 0.31824344396591187
```

For the Word - "Thriller"

```
: search_query = 'thriller'
if search_query in model.wv.key_to_index:
    similar_words = model.wv.most_similar(search_query, topn = 10)
    print(f"\nWords similar to '{search_query}' - ")
    for word, similarity in similar_words:
        print(f"{word} - {similarity}")
else:
    print(f"\n'{search_query}' is not in the vocabulary.")
```

```
Words similar to 'thriller' -
conduct - 0.45576658844947815
cares - 0.37544986605644226
charts - 0.3489747941493988
goh - 0.33104124665260315
fields - 0.3287579119205475
fin - 0.3199218511581421
comical - 0.31686532497406006
signs - 0.31665536761283875
efsun - 0.3160402774810791
honors - 0.31176427006721497
```

For the Word - "Drama"

```
search_query = 'drama'
if search_query in model.wv.key_to_index:
    similar_words = model.wv.most_similar(search_query, topn = 10)
    print(f"\nWords similar to '{search_query}' - ")
    for word, similarity in similar_words:
        print(f"{word} - {similarity}")
else:
    print(f"\n'{search_query}' is not in the vocabulary.")
```

```
Words similar to 'drama' -
find - 0.4521694779396057
two - 0.4139154553413391
man - 0.4017150104045868
one - 0.39718112349510193
assumes - 0.39636096358299255
ajegunle - 0.3950473368167877
wife - 0.39069730043411255
crime - 0.38538020849227905
unexpected - 0.3821523189544678
lawyer - 0.37965816259384155
```

Explanation-

In the code provided above we performed Sentiment Analysis for the words in the 'description' column and made a Similarity Search for the words - Love, Animated, Thriller and Drama.

Preprocessing of the data

Here, we have taken the Netflix dataset and loaded it. In the preprocessing step, we wanted to remove the stop words as they don't add any value for the sentiment analysis as they are neutral and later converted the data to the lowercase.

Trained the model and Evaluated the results

Later, we trained the model using the word2vec function and were able to get the sentiment analysis scores. We added a new column named "label" to the data frame with the labelled data.

Every name in the "name" column was looked at to determine whether the words "excellent" or "great" or other similar positive words (in lowercase) were used anywhere. If the positive word occurs in the name, we give it a label of 1 (expressing positivity); and all the positive words and similarly, -1 for the negative words.

And finally applied the word2vec functionality to find the sentiment scores for the description column

For the Similarity Search of the words - Drama, Thriller, Animated, Love using the 'most similar' functionality of the word2_vec model to obtain the top 10 similar words for the given word.

References

- <https://sparkbyexamples.com/spark/spark-word-count-example>
- <https://www.tensorflow.org/tutorials/text/word2vec#:~:text=word2vec%20is%20not%20a%20singular,downstream%20natural%20language%20processing%20tasks>.
- <https://medium.com/swlh/sentiment-classification-using-word-embeddings-word2vec-aedf28fbb8ca>
- <https://www.districtdatalabs.com/modern-methods-for-sentiment-analysis>