1. Functional Programming [25 points]

1. Create a function add that takes an arbitrary number of arguments, and adds them all. Also create a function sub that subtracts all the arguments but the first from the first1. Also create a function ra_sub that performs right-associative subtraction add(1, 2, 3) => 6 sub(5, 1, 2) => 2 ra_sub(5, 1, 2) => (5 - (1 - 2)) => 6

```
from functools import reduce

# Function to add arbitrary number of arguments
def add(*args):
    return sum(args)

# Function to subtract all arguments but the first from the first
def sub(first, *args):
    return first - sum(args)

# Function to perform right-associative subtraction
def ra_sub(*args):
    return reduce(lambda x, y: y - x, reversed(args))

# Test cases
print(add(1, 2, 3))  # Output: 6
print(sub(5, 1, 2))  # Output: 2
print(ra_sub(5, 1, 2))  # Output: 6
```

```
6
2
6
```

1. Create a function zip that takes an arbitrary number of sequences, and zips them, i.e. creates a list of lists, where the inner lists consist of the first elements from the given sequences, then the second elements from the given sequences, and so on.

   zip([1, 2, 3], [4, 5, 6]) => [[1, 4], [2, 5], [3, 6]]

   zip([1, 2, 3], [4, 5, 6], [7, 8, 9]) => [[1, 4, 7], [2, 5, 8], [3, 6, 9]]

```
# Function to zip multiple sequences into a list of lists
def zip_lists(*sequences):
    return list(map(list, zip(*sequences)))

# Test cases
print(zip_lists([1, 2, 3], [4, 5, 6]))  # Output: [[1, 4], [2, 5], [3,
6]]
print(zip_lists([1, 2, 3], [4, 5, 6], [7, 8, 9]))  # Output: [[1, 4,
7], [2, 5, 8], [3, 6, 9]]
```

```
[[1, 4], [2, 5], [3, 6]]
[[1, 4, 7], [2, 5, 8], [3, 6, 9]]
```

1. Create a function zipwith that takes a function f and an arbitrary number of sequences, and returns a list of f applied to the first elements of the given sequences, followed by f applied to the second elements of the sequences, and so on.

   zipwith(add, [1, 2, 3], [4, 5, 6]) => [5, 7, 9]

   zipwith(add, [1, 2, 3], [4, 5, 6], [1, 1, 1]) => [6, 8, 10]

```
from functools import partial

# Function to apply a given function on elements of multiple sequences
def zipwith(f, *sequences):
    return list(map(lambda *args: f(*args), *sequences))

# Test cases
print(zipwith(add, [1, 2, 3], [4, 5, 6]))  # Output: [5, 7, 9]
print(zipwith(add, [1, 2, 3], [4, 5, 6], [1, 1, 1]))  # Output: [6, 8,
10]

[5, 7, 9]
[6, 8, 10]
```

1. Create a function flatten that can flatten a tree.

   flatten([1, [2, [3, 4], [5, 6], 7], 8, [9, 10]]) => [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

```
# Function to flatten nested lists (or trees)
def flatten(lst):
   return sum(map(flatten, lst), []) if isinstance(lst, list) else
[lst]

# Test case
print(flatten([1, [2, [3, 4], [5, 6], 7], 8, [9, 10]]))  # Output: [1,
2, 3, 4, 5, 6, 7, 8, 9, 10]

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

1. Create a function group_by that takes a function and a sequence and groups the elements of the sequence based on the result of the given function. In the example below, len returns the length of a sequence.

   group_by(len, ["hi", "dog", "me", "bad", "good"]) => {2: ["hi", "me"], 3: ["dog", "bad"], 4: ["good"]}

```
from collections import defaultdict
from functools import reduce
```

```
# Function to group elements of a sequence based on a given function
def group_by(f, sequence):
    return reduce(lambda acc, item: {**acc, f(item): acc.get(f(item),
[]) + [item]}, sequence, defaultdict(list))

# Test case
print(group_by(len, ["hi", "dog", "me", "bad", "good"]))  # Output:
{2: ['hi', 'me'], 3: ['dog', 'bad'], 4: ['good']}
```

```
{2: ['hi', 'me'], 3: ['dog', 'bad'], 4: ['good']}
```

2. Confirming Hadoop Installation [15 points]

1. [3 points] Acquire the cluster.

I followed step one in the big-data-repo github page belonging to Professor Singh. I acquired the cluster doing those steps plus disabling internal ip address option.

1. [3 points] Load the data into the master, move the data into HDFS.

I sshed into the cluster master and cloned the repo and then created a directory called /user/adityad and put the files from the cloned repo into the HDFS and then checked if the five-books are present there.

Output:

Found 5 items

-rw-r--r-- 1 aditya_duggirala hadoop 179903 2024-10-13 19:13 /user/adityad/five-books/a_tangled_tale.txt

-rw-r--r-- 1 aditya_duggirala hadoop 173379 2024-10-13 19:13 /user/adityad/five-books/alice_in_wonderland.txt

-rw-r--r-- 1 aditya_duggirala hadoop 394246 2024-10-13 19:13 /user/adityad/five-books/sylvie_and_bruno.txt

-rw-r--r-- 1 aditya_duggirala hadoop 458755 2024-10-13 19:13 /user/adityad/five-books/symbolic_logic.txt

-rw-r--r-- 1 aditya_duggirala hadoop 135443 2024-10-13 19:13 /user/adityad/five-books/the_game_of_logic.txt

1. [3 points] Without writing any code of your own, verify that you have a good installation of hadoop by running wordcount on five-books. The command is similar to

   hadoop jar /usr/lib/hadoop-mapreduce/hadoop-mapreduce-examples.jar wordcount /user/singhj/five-books /books-count

I ran the command and this is the ouput I got:

2024-10-13 22:23:30,287 INFO mapreduce.Job: Job job_1728858072744_0001 completed successfully 2024-10-13 22:23:30,376 INFO mapreduce.Job: Counters: 55 File System Counters FILE: Number of bytes read=596684 FILE: Number of bytes written=3493255 FILE: Number of read operations=0 FILE: Number of large read operations=0 FILE: Number of write operations=0 HDFS: Number of bytes read=1342381 HDFS: Number of bytes written=313829 HDFS: Number of read operations=30 HDFS: Number of large read operations=0 HDFS: Number of write operations=9 HDFS: Number of bytes read erasure-coded=0 Job Counters Killed reduce tasks=1 Launched map tasks=5 Launched reduce tasks=3 Data-local map tasks=5 Total time spent by all maps in occupied slots (ms)=163929804 Total time spent by all reduces in occupied slots (ms)=97618380 Total time spent by all map tasks (ms)=48414 Total time spent by all reduce tasks (ms)=28830 Total vcore-milliseconds taken by all map tasks=48414 Total vcore-milliseconds taken by all reduce tasks=28830 Total megabyte-milliseconds taken by all map tasks=163929804 Total megabyte-milliseconds taken by all reduce tasks=97618380 Map-Reduce Framework Map input records=35119 Map output records=219095 Map output bytes=2091576 Map output materialized bytes=596756 Input split bytes=655 Combine input records=219095 Combine output records=42051 Reduce input groups=29287 Reduce shuffle bytes=596756 Reduce input records=42051 Reduce output records=29287 Spilled Records=84102 Shuffled Maps =15 Failed Shuffles=0 Merged Map outputs=15 GC time elapsed (ms)=433 CPU time spent (ms)=13200 Physical memory (bytes) snapshot=4379766784 Virtual memory (bytes) snapshot=38611079168 Total committed heap usage (bytes)=4391436288 Peak Map Physical memory (bytes)=622166016 Peak Map Virtual memory (bytes)=4832083968 Peak Reduce Physical memory (bytes)=466513920 Peak Reduce Virtual memory (bytes)=4836503552 Shuffle Errors BAD_ID=0 CONNECTION=0 IO_ERROR=0 WRONG_LENGTH=0 WRONG_MAP=0 WRONG_REDUCE=0 File Input Format Counters Bytes Read=1341726 File Output Format Counters Bytes Written=313829

1. [3 points] Run wordcount using the provided mapper_noll.py and the default reducer aggregate. The command is similar to

   mapred streaming -file ~/big-data-repo/hadoop/mapper_noll.py -mapper mapper_noll.py -input /user/singhj/five-books -reducer aggregate -output /books-stream-count

I ran the command and this is the output:

2024-10-13 22:32:23,447 INFO mapreduce.Job: Job job_1728858072744_0002 completed successfully 2024-10-13 22:32:23,554 INFO mapreduce.Job: Counters: 54 File System Counters FILE: Number of bytes read=660527 FILE: Number of bytes written=5387912 FILE: Number of read operations=0 FILE: Number of large read operations=0 FILE: Number of write operations=0 HDFS: Number of bytes read=1367597 HDFS: Number of bytes written=103696 HDFS: Number of read operations=48 HDFS: Number of large read operations=0 HDFS: Number of write operations=9 HDFS: Number of bytes read erasure-coded=0 Job Counters Launched map tasks=11 Launched reduce tasks=3 Data-local map tasks=11 Total time spent by all maps in occupied slots (ms)=358150764 Total time spent by all reduces in occupied slots (ms)=93050666 Total time spent by all map tasks (ms)=105774 Total time spent by all reduce tasks (ms)=27481 Total vcore-milliseconds taken by all map tasks=105774 Total vcore-milliseconds taken by all reduce tasks=27481 Total megabyte-milliseconds taken by all map tasks=358150764 Total megabyte-milliseconds taken by all reduce tasks=93050666 Map-

Reduce Framework Map input records=35119 Map output records=207438 Map output bytes=4167234 Map output materialized bytes=660707 Input split bytes=1295 Combine input records=207438 Combine output records=26905 Reduce input groups=10201 Reduce shuffle bytes=660707 Reduce input records=26905 Reduce output records=10201 Spilled Records=53810 Shuffled Maps =33 Failed Shuffles=0 Merged Map outputs=33 GC time elapsed (ms)=878 CPU time spent (ms)=20770 Physical memory (bytes) snapshot=7693291520 Virtual memory (bytes) snapshot=67552075776 Total committed heap usage (bytes)=8057257984 Peak Map Physical memory (bytes)=643846144 Peak Map Virtual memory (bytes)=4834713600 Peak Reduce Physical memory (bytes)=429309952 Peak Reduce Virtual memory (bytes)=4830900224 Shuffle Errors BAD_ID=0 CONNECTION=0 IO_ERROR=0 WRONG_LENGTH=0 WRONG_MAP=0 WRONG_REDUCE=0 File Input Format Counters Bytes Read=1366302 File Output Format Counters Bytes Written=103696 2024-10-13 22:32:23,555 INFO streaming.StreamJob: Output directory: /books-stream-count

1. [3 points] Run wordcount using the provided mapper_noll.py and the provided reducer reducer_noll.py. The command is similar to

   mapred streaming -files ~/big-data-repo/hadoop/mapper_noll.py ~/big-data-repo/hadoop/reducer_noll.py -mapper mapper_noll.py -reducer reducer_noll.py -input /user/singhj/five-books -output /books-my-own-counts

I had to ran this command instead since the one in question 5 of this section didn't work due to an extra argument.

This is the command I ran:

mapred streaming -file ~/big-data-repo/hadoop/mapper_noll.py -mapper mapper_noll.py -file ~/big-data- repo/hadoop/reducer_noll.py -reducer reducer_noll.py -input / user/adityad/five-books -output /books-my-own-counts

This is the output:

2024-10-13 22:43:07,569 INFO mapreduce.Job: Job job_1728858072744_0003 completed successfully 2024-10-13 22:43:07,652 INFO mapreduce.Job: Counters: 55 File System Counters FILE: Number of bytes read=4582128 FILE: Number of bytes written=13240172 FILE: Number of read operations=0 FILE: Number of large read operations=0 FILE: Number of write operations=0 HDFS: Number of bytes read=1367597 HDFS: Number of bytes written=236309 HDFS: Number of read operations=48 HDFS: Number of large read operations=0 HDFS: Number of write operations=9 HDFS: Number of bytes read erasure-coded=0 Job Counters Killed reduce tasks=1 Launched map tasks=11 Launched reduce tasks=3 Data-local map tasks=11 Total time spent by all maps in occupied slots (ms)=358990492 Total time spent by all reduces in occupied slots (ms)=101329436 Total time spent by all map tasks (ms)=106022 Total time spent by all reduce tasks (ms)=29926 Total vcore-milliseconds taken by all map tasks=106022 Total vcore-milliseconds taken by all reduce tasks=29926 Total megabyte-milliseconds taken by all map tasks=358990492 Total megabyte-milliseconds taken by all reduce tasks=101329436 Map-Reduce Framework Map input records=35119 Map output records=207438 Map output bytes=4167234 Map output materialized bytes=4582308 Input split bytes=1295 Combine input records=0 Combine output records=0 Reduce input groups=10201 Reduce shuffle bytes=4582308 Reduce input records=207438 Reduce output records=10201 Spilled Records=414876 Shuffled Maps =33 Failed Shuffles=0 Merged Map outputs=33 GC time

elapsed (ms)=955 CPU time spent (ms)=21370 Physical memory (bytes) snapshot=7863332864 Virtual memory (bytes) snapshot=67558744064 Total committed heap usage (bytes)=8215592960 Peak Map Physical memory (bytes)=630722560 Peak Map Virtual memory (bytes)=4831543296 Peak Reduce Physical memory (bytes)=497270784 Peak Reduce Virtual memory (bytes)=4831469568 Shuffle Errors BAD_ID=0 CONNECTION=0 IO_ERROR=0 WRONG_LENGTH=0 WRONG_MAP=0 WRONG_REDUCE=0 File Input Format Counters Bytes Read=1366302 File Output Format Counters Bytes Written=236309 2024-10-13 22:43:07,652 INFO streaming.StreamJob: Output directory: /books-my-own-counts

3. Analyzing Server Logs [55 points]

A dataset representing Apache web server logs is available as access.log. Each row has the following schema:

● IP of client: This refers to the IP address of the client that sent the request to the server.

● Remote Log Name: Remote name of the User performing the request. In the majority of the applications, this is confidential information and is hidden or not available.

● User ID: The ID of the user performing the request. In the majority of the applications, this is a piece of confidential information and is hidden or not available.

● Date and Time: The date and time of the request are represented in UTC format as follows:

-Day/Month/Year:Hour:Minutes: Seconds +Time-Zone-Correction.

● Request Type: The type of request (GET, PUT, POST, etc.) that the server got. This depends on the operation that the request will do.

● API: The API of the website to which the request is related. Example: When a user accesses a cart on a shopping website, the API comes as /usr/cart.

● Protocol and Version: Protocol used for connecting with server and its version.

● Status Code: Status code that the server returned for the request. Eg: 404 is sent when a requested resource is not found. 200 is sent when the request was successfully served. See the http status code registry listing for interpretations of status codes.

● Byte: The amount of data in bytes that was sent back to the client.

● Referrer: The websites/source from where the user was directed to the current website. If none, it is represented by “-“.

● User Agent String: The user agent string contains details of the browser and the host device (like the name, version, device type etc.).

● Response Time: The response time the server took to serve the request. This is the difference between the timestamps when the request was received and when the request was served.

Use Hadoop to perform analytics on the provided data3.

1. [6+9=15 points4] What is the percentage of each request type (GET, PUT, POST, etc.)?

I first figured out how many request types there are. Then I wanted to know what those request types are. Finally, I wanted to find the unique count of each request type.

I did these commands:

aditya_duggirala@cluster-a30e-m:~/big-data-repo/datasets$ grep -oP '"(GET|POST|PUT|DELETE|HEAD|OPTIONS|PATCH) ' access.log | sort | uniq | wc -l

3

aditya_duggirala@cluster-a30e-m:~/big-data-repo/datasets$ grep -oP '"(GET|POST|PUT|DELETE|HEAD|OPTIONS|PATCH) ' access.log | sort | uniq

"GET

"HEAD

"POST

aditya_duggirala@cluster-a30e-m:~/big-data-repo/datasets$ grep -oP '"(GET|POST|PUT|DELETE|HEAD|OPTIONS|PATCH) ' access.log | sort | uniq -c

33414 "GET

253 "HEAD

44584 "POST

The percentages are:

GET: 42.71%

HEAD: 0.32%

POST: 56.98%

1. [6+9=15 points] What percent of the responses fall into each of the following five types?

   - Informational responses (100–199)
   - Successful responses (200–299)
   - Redirection messages (300–399)
   - Client error responses (400–499)
   - Server error responses (500–599)

I used awk to process each line of text in the access log and print the 9th field which is the status code paired with grep to search for extended regular expression pattern of 100-199 200-299 and etc. Finally, I got the line count of each status code.

I did these commands:

aditya_duggirala@cluster-a30e-m:~/big-data-repo/datasets$

#Count informational responses (100–199) awk '{print $9}' access.log | grep -E '1[0-9][0-9]' | wc -l

#Count successful responses (200–299) awk '{print $9}' access.log | grep -E '2[0-9][0-9]' | wc -l

#Count redirection messages (300–399) awk '{print $9}' access.log | grep -E '3[0-9][0-9]' | wc -l

#Count client error responses (400–499) awk '{print $9}' access.log | grep -E '4[0-9][0-9]' | wc -l

#Count server error responses (500–599) awk '{print $9}' access.log | grep -E '5[0-9][0-9]' | wc -l

0

70684

2929

4638

0

The percentages are:

(100–199): 0%

(200–299): 90.33%

(300–399): 3.74%

(400–499): 5.93%

(500–599): 0%

1. [9+16=25 points] What 5 IP addresses generate the most client errors?

I used awk to extract the 1st field and 9th field to get the IP-Addresses and status code and then filter the lines to include the client error status codes 400-499 and then extract the ip-addresses with those codes. Finally I sorted the top five unique counted ip address from the highest count to lowest count of client errors

I used this command:

awk print $1,9$access.log \vee grep '4[0-9][0-9]' | awk '{print $1}' | sort | uniq -c | sort -nr | head -n 5

2059 173.255.176.5

126 212.9.160.24

78 13.77.204.88

58 51.210.243.185

53 193.106.30.100

4. Presidential Speeches [15 points]

All US presidential speeches are available as a single zip in J's Github repo. Each speech may be cleaned with this filter:

import requests

import re

import string

stopwords_list = requests.get("https://gist.githubusercontent.com/rg089/35e00abf8941d72d419224cfd5b5925d/raw/12d899b70156fd0041fa9778d657330b024b959c/stopwords.txt").content stopwords = list(set(stopwords_list.decode().splitlines()))

def remove_stopwords(words): list_ = re.sub(r"[^a-zA-Z0-9]", " ", words.lower()).split()) return [itm for itm in list_ if itm not in stopwords]

def clean_text(text): text = text.lower() text = re.sub('[.*?]', '', text) text = re.sub('[%s]' % re.escape(string.punctuation), ' ', text) text = re.sub('[\d\n]', ' ', text) return ' '.join(remove_stopwords(text))

The goal of this analysis is to calculate the sentiment of each president's speeches. One way to compute the sentiment of a collection of words is to take the average of their valences.5 The AFINN-165 collection contains the valences of 3,382 English words.

Write a function valence (text) such that it takes a line of any presidential speech and returns its valence after cleaning it. It should be a functional program, conforming to the pattern:

def valence(text):

return calc_valence(clean_text(text))

where calc_valence(text) is a function that you write. Be sure to test this function under any imaginable conditions, for example:

- When text is empty,
- When text is a string of non-printable characters,
- When text is a bytecode string,

The function must be in a form that we can use for testing in our environment against data that you don't have access to. The presidential speeches should be considered a representative sample.

Compute the average valence of each president's speeches according to this outline:

1. [7 points] In the mapper (which is given a sequence of lines of speeches as input):

a. Clean each line as suggested above,

b. Calculate the valence of each word in the line,

c. Emit a (tab-separated) key-value pair (president, word valence) for each word in the line.

2. [6 points] In the reducer (which is given all (president, word valence) key-value pairs with the same key, i.e.president):

   a. Compute the average valence of all words spoken by the president,

   b. Emit a (tab-separated) key-value pair (president, sentiment of president's speeches).

For both of these questions I have a txt files for these that I will include in the zip folder of quiz 4.

```
#!/usr/bin/env



import re
import string
import requests

# Download stopwords list
stopwords_url =
"https://gist.githubusercontent.com/rg089/35e00abf8941d72d419224cfd5b5
925d/raw/12d899b70156fd0041fa9778d657330b024b959c/stopwords.txt"
stopwords_list = requests.get(stopwords_url).content
stopwords = set(stopwords_list.decode().splitlines())

# Function to remove stopwords and clean text
def remove_stopwords(words):
    list_ = re.sub(r"[^a-zA-Z0-9]", " ", words.lower()).split()  #
Remove non-alphanumeric characters
    return [itm for itm in list_ if itm not in stopwords]

# Function to clean text and remove stopwords
def clean_text(text):
    text = text.lower()
    text = re.sub(r'\[.*?\]', '', text)  # Remove anything in brackets
    text = re.sub(r'[%s]' % re.escape(string.punctuation), ' ', text)
# Remove punctuation
    text = re.sub(r'[\d\n]', ' ', text)  # Remove digits and newlines
    return ' '.join(remove_stopwords(text))


# Load the AFINN wordlist (replace the path with your correct path)
afinn = {}
afinn_path = "C:\\Users\\adity\\OneDrive\\Tufts University Online MS
```

```
CS\\2024\\Fall Sem\\CS 119\\Quizzes\\Quiz4\\AFINN-en-165.txt"

with open(afinn_path, 'r') as file:
    for line in file:
        word, score = line.split('\t')
        afinn[word] = int(score)

# Function to calculate the valence
def calc_valence(text):
    if isinstance(text, bytes):
        text = text.decode("utf-8", errors="ignore")

    if not text.strip():
        return 0

    words = text.split()
    valence_sum = 0
    word_count = 0

    for word in words:
        if word in afinn:
            valence_sum += afinn[word]
            word_count += 1

    return valence_sum / word_count if word_count > 0 else 0

# Main valence function
def valence(text):
    if isinstance(text, bytes):  # Convert bytecode to string
        text = text.decode("utf-8", errors="ignore")
    cleaned_text = clean_text(text)
    return calc_valence(cleaned_text)

# Testing the valence function
print(valence(""))  # Edge case: empty text
print(valence("!@#$%"))  # Edge case: special characters only
print(valence(b"\x80\x81\x82"))  # Edge case: bytecode strings

# Print a sample of AFINN words and their valence scores
print("Sample of AFINN words and their valence scores:",
list(afinn.items())[:10])

print(valence("happy joy love"))  # Should return a positive valence
print(valence("sad anger hate"))  # Should return a negative valence

0
0
0
Sample of AFINN words and their valence scores: [('abandon', -2),
('abandoned', -2), ('abandons', -2), ('abducted', -2), ('abduction', -
```

```
2), ('abductions', -2), ('abhor', -3), ('abhorred', -3), ('abhorrent',
-3), ('abhors', -3)]
3.0
-2.6666666666666665
```

[2 points] How much data, in bytes, was emitted by the mappers?

Map-Reduce Framework

Map input records=40232

Map output records=39210

Map output bytes=596825

596,825 bytes were emitted.

5. Hadoop Errors [15 points]

When dealing with errors in Hadoop, where the execution is distributed to hundreds of workers, an error message could end up in a log file on any of those servers. This is a scavenger hunt question. We deliberately modify the code so it would occasionally fail and look for the error message so we can find them! The provided mapper for Hadoop Streaming, mapper_noll.py, with the changed lines shown in red7. Run Hadoop Streaming on the five books we have been using for practice, using this modified mapper. It will fail, of course!

[7 points] Where (what server & location) did the divide-by-zero error messages show up and how many did you find?

[8 points] How many such messages did you find? Is the count you found consistent with what you might expect from random.randint(0,99)?

Explainations for both questions below:

Start Hadoop Streaming Job:

First, I modified the mapper_noll.py file by uncommenting the line responsible for generating the divide-by-zero error:

nano ~/big-data-repo/hadoop/mapper_noll.py I uncommented the line:

x = 1 / random.randint(0,99) Then, I used the following command to run the Hadoop Streaming job:

mapred streaming -file ~/big-data-repo/hadoop/mapper_noll.py -mapper mapper_noll.py -input /user/adityad/five-books -reducer -output /hadoop-error-test The job failed due to the divide-by-zero error.

Find the Application ID:

To diagnose the issue, I listed all YARN applications to find the application ID of the failed job:
bash

yarn application -list -appStates ALL

I found the application ID as application_1728922376681_0001, which I would use in subsequent steps. Extract Logs for the Application:

Using the application ID, I fetched the logs to search for divide-by-zero errors:

yarn logs -applicationId application_1728922376681_0001 | grep "ZeroDivisionError"

Since the logs didn't explicitly mention "ZeroDivisionError,"

I looked for other related errors, specifically checking for failures due to subprocess failed with code 1:

yarn logs -applicationId application_1728922376681_0001 | grep "subprocess failed with code 1"

Server and Location:

The server responsible for processing this job is cluster-a30e-m.c.united-bot-438500-e7.internal, as shown in the logs:

Connecting to ResourceManager at
cluster-a30e-m.c.united-bot-438500-e7.internal./10.128.0.2:8032

The location where the divide-by-zero errors occurred is within the tasks executed on this Hadoop cluster, as evidenced by the repeated task failure reports in the logs.

Count of IPC Server Handlers:

I counted how many times the divide-by-zero error occurred by looking for IPC Server handler messages, which are involved when tasks fail due to the mapper's error:

yarn logs -applicationId application_1728922376681_0001 | grep "IPC Server handler" | wc -l

This command returned 90 IPC handler logs indicating task failures due to the divide-by-zero error.

Expected Outcome Based on random.randint(0,99) Logic:

The code random.randint(0,99) has a 1% chance of generating a zero, which triggers the divide-by-zero error. Given that the job launched many tasks and each task processes multiple lines of input, having 90 errors is consistent with this probability. Since the job involved launching multiple map tasks, with each task processing a portion of the input data, the total number of errors (90) falls within the expected range for a 1% error rate.