CSCI 5523 Introduction to Data Mining

Spring 2025

Assignment 1 (10 points)

Deadline: February. 24th 11:59 PM CDT

1. Overview of the Assignment

In assignment 1, you will complete three tasks. The goal of these tasks is to help you get familiar with Spark operations (e.g., transformations and actions) and MapReduce.

2. Requirements

2.1 Programming Requirements

a. You must use Python to implement all tasks. You can only use standard Python libraries (i.e., you may use libraries like math, but external libraries like numpy or pandas are not allowed). If you have questions about specific packages, please ask on Piazza.
b. **You are required to only use Spark RDD**, i.e., there is no point in using Spark DataFrame or DataSet.

2.2 Submission Platform

We will use Gradescope to run and grade your submission automatically. We highly recommend that you first test your scripts on your local machine before submitting your solutions.

2.3 Programming Environment

Python 3.9.12 and Spark 3.2.1

You will need to have Java installed to run PySpark. We tested the environment with JDK 11. Please note that PySpark requires Java 8 (but will not work with versions prior to update 8u371), 11 or 17.

We recommend using either Miniconda or Anaconda to create a virtual environment to run your code in. To install miniconda or anaconda, you may reference the following links: https://docs.anaconda.com/miniconda/install/ and https://docs.anaconda.com/anaconda/install/

Once conda is installed you can reference the following link to install pyspark: https://spark.apache.org/docs/latest/api/python/getting_started/install.html#using-conda

2.4 Write Your Own Code

Do not use large language models (e.g., ChatGPT, Copilot) or share code with other students!

For this assignment to be an effective learning experience, **you must write your own code**. We emphasize this point because you will be able to find Python implementations of some of the required functions on the web. Please do not look for or at any such code.

TAs will combine all the code we can find from the web (e.g., Github), other students' code from this and other (previous) sections, and code generated from AI tools for plagiarism detection. **We will report all detected plagiarism to the university.**

3. Yelp Data

In this assignment, you are provided with two datasets (reviews.json and business.json) extracted from the Yelp dataset [https://www.yelp.com/dataset] for developing your assignment. You can access and download the datasets from the "data" folder on Google Drive. **We generated these datasets in a random sampling way. These given datasets are only for your testing. During the grading period, we will use different sampled subsets of the Yelp datasets.**

4. Tasks

You need to submit the following files (all lowercase):

Python scripts: `task1.py`, `task2.py`, `task3_default.py`, `task3_customized.py`
PDF file: `task3.pdf`

4.1 Task1: Data Exploration (5 points)

4.1.1 Task description

You will explore the **review dataset** and write a program that computes the answer to the following questions:
A. What is the number of distinct businesses?
B. What is the number of distinct users who have written more than ***m*** reviews?
C. What is the total review count for each year?
D. What is the number of reviews with star rating ***s***?
E. What are the top ***i*** frequent words in the review text for a given year ***y***?

Preprocessing:

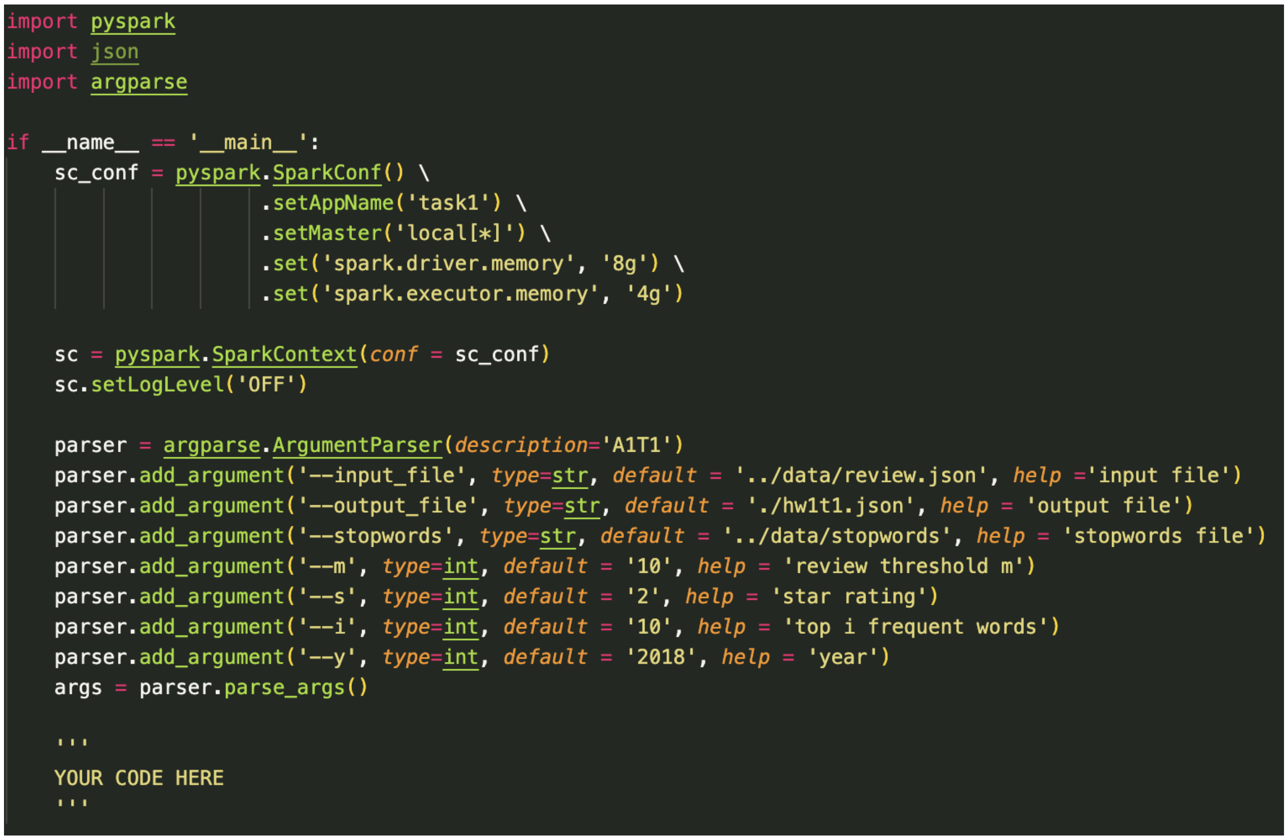
1. The words should be **in lowercase**.
2. Remove the following punctuation: "(", "[", ",", ".", "!", "?", ":", ";", "]", ")", if they are leading or trailing characters in your words.
3. You should also exclude the given stopwords. You can access the file that contains stopwords from the "data" folder.

- Example: "Husband, daughter & I got there at 2:30, and food is amazing !!!   (5 stars!!),"
  Stopwords: ['I', 'there', 'at', 'and', 'is']
  After preprocessing: ['husband', 'daughter', '&', 'got', '2:30', 'food', 'amazing', '5', 'stars']

4.1.2 Execution commands

We have provided the skeleton code pictured below (Figure 1) in the `task1.py` file. Please do not change the name of any of the command line parameters. Feel free to write your own helper functions.

You may also import packages as long as they are default Python packages (see Section 2.1).



```python
import pyspark
import json
import argparse

if __name__ == '__main__':
    sc_conf = pyspark.SparkConf() \
                     .setAppName('task1') \
                     .setMaster('local[*]') \
                     .set('spark.driver.memory', '8g') \
                     .set('spark.executor.memory', '4g')

    sc = pyspark.SparkContext(conf = sc_conf)
    sc.setLogLevel('OFF')

    parser = argparse.ArgumentParser(description='A1T1')
    parser.add_argument('--input_file', type=str, default = '../data/review.json', help ='input file')
    parser.add_argument('--output_file', type=str, default = './hw1t1.json', help = 'output file')
    parser.add_argument('--stopwords', type=str, default = '../data/stopwords', help = 'stopwords file')
    parser.add_argument('--m', type=int, default = '10', help = 'review threshold m')
    parser.add_argument('--s', type=int, default = '2', help = 'star rating')
    parser.add_argument('--i', type=int, default = '10', help = 'top i frequent words')
    parser.add_argument('--y', type=int, default = '2018', help = 'year')
    args = parser.parse_args()

    '''
    YOUR CODE HERE
    '''
```

Figure 1. Skeleton code for task1.

You can execute **`task1.py`** using the following command line:

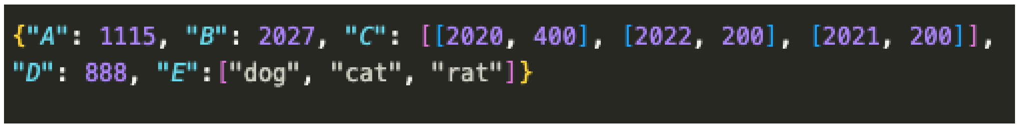
```
Python
python task1.py --input_file <input_file> --output_file <output_file>
--stopwords <stopwords> --m <m> --s <s> --i <i> --y <y>
```

Params:

`input_file` – the input file (the review dataset)
`output_file` – the output file contains your answers
`stopwords` – the file contains the stopwords that should be removed for E
`m/s/i/y` – see 4.1.1

4.1.3 Output format:

You must write the results in the JSON format using **exactly the same tags** for each question (see an example in Figure 2). You must also follow the format in Figure 2. The answer for A/B/D is a number. The answer for C is a list of pairs **[year, count]** sorted by the count in the descending order. If two years have the same count, please sort them in descending numerical order of the year (i.e. bigger numbers first). The answer for E is a list of frequent words sorted in descending order from most to least frequent. If two words have the same frequency then sort them lexicographically.



```
{"A": 1115, "B": 2027, "C": [[2020, 400], [2022, 200], [2021, 200]],
"D": 888, "E":["dog", "cat", "rat"]}
```

Figure 2. An example output for task1 in JSON format

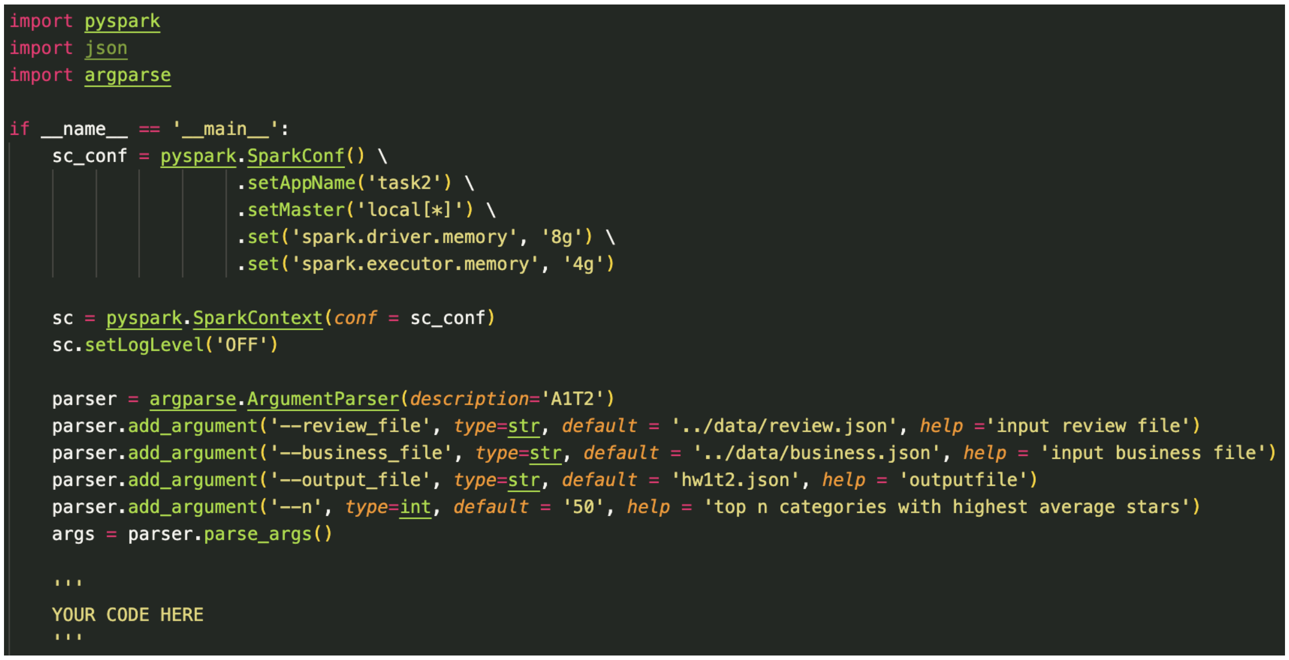
4.2 Task2: Exploration on Multiple Datasets (2 points)

4.2.1 Task description

In task2, you will explore the two datasets together (**review and business datasets**). Write a program to find the top n categories with the highest average star rating. Some businesses will have multiple entries under the 'categories' tag. In this case the star rating for that business should contribute to the average for each of those categories. Some businesses will also have None under their categories tag, in which case you should filter them out.

4.2.2 Execution commands (using argparse as 4.1.2)

We have provided the skeleton code pictured below (Figure 3) in the **`task2.py`** file. Please do not change the name of any of the command line parameters. Feel free to write your own helper functions. You may also import packages as long as they are default Python packages (see Section 2.1).



```
import pyspark
import json
import argparse

if __name__ == '__main__':
    sc_conf = pyspark.SparkConf() \
                    .setAppName('task2') \
                    .setMaster('local[*]') \
                    .set('spark.driver.memory', '8g') \
                    .set('spark.executor.memory', '4g')

    sc = pyspark.SparkContext(conf = sc_conf)
    sc.setLogLevel('OFF')

    parser = argparse.ArgumentParser(description='A1T2')
    parser.add_argument('--review_file', type=str, default = '../data/review.json', help ='input review file')
    parser.add_argument('--business_file', type=str, default = '../data/business.json', help = 'input business file')
    parser.add_argument('--output_file', type=str, default = 'hw1t2.json', help = 'outputfile')
    parser.add_argument('--n', type=int, default = '50', help = 'top n categories with highest average stars')
    args = parser.parse_args()

    '''
    YOUR CODE HERE
    '''
```

Figure 3. Skeleton code for task2.

Python

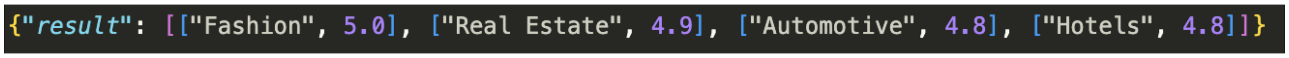
```
python task2.py --review_file <review_file> --business_file <business_file>
--output_file <output_file> --n <n>
```

Params:

- **`review_file`** – the input file (the review dataset)
- **`business_file`** – the input file (the business dataset)
- **`output_file`** – the output file contains your answers
- **n** – top n categories with highest average stars (see 4.2.1)

4.2.3 Output format:

You must write the results in a json file following the same format as Figure 4. The answer is a list of pairs **[category, ave_star_rating]**, which are sorted by the average star rating in descending order. If two categories have the same value, please sort the categories in the alphabetical order.



```
{"result": [["Fashion", 5.0], ["Real Estate", 4.9], ["Automotive", 4.8], ["Hotels", 4.8]]}
```

Figure 4: An example output for task2 in JSON format

4.3 Task3: Partition (3 points)

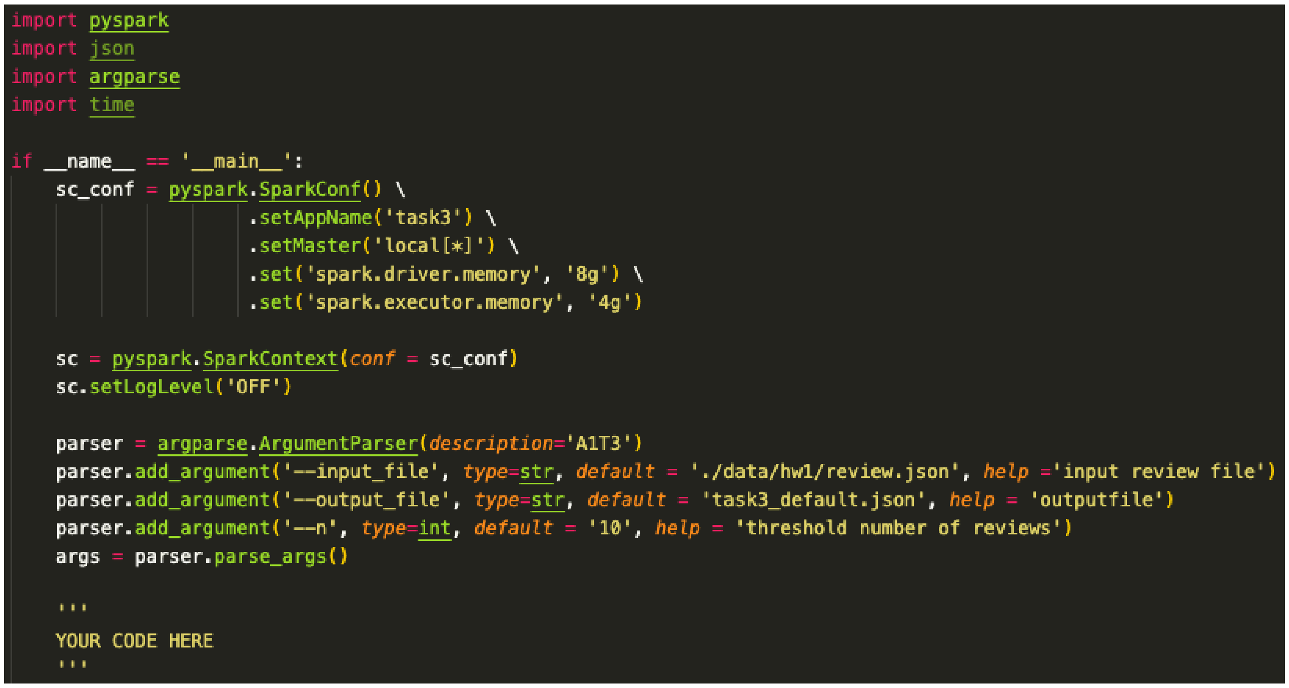
4.3.1 Task description

In this task, you will learn how partitions work in the RDD. You need to use the **review dataset** and compute the reviewers that reviewed more than **n** businesses **in the review file**. You will return a list of (user, review_count) pairs for all users that pass the threshold **n**.

You will also need to show the number of partitions for the RDD and the number of items per partition. You will implement two partition functions: **(1) the default partition function with the default partition number; (2) a customized partition function with an input n_partitions** i.e., design a customized partition function (like a hash function of your choice). You need to describe the customized partition function you use, compare the execution time of the two methods (default vs. customized), and examine how the number of partitions affect the execution time. You need to justify the result with one or two sentences (write your answer in a PDF file “task3.pdf”).

4.3.2 Execution commands

We have provided the skeleton code pictured below in the **`task3_default.py`** file. Please do not change the name of any of the command line parameters. Feel free to write your own helper functions. You may also import packages as long as they are default Python packages (see Section 2.1).



```python
import pyspark
import json
import argparse
import time

if __name__ == '__main__':
    sc_conf = pyspark.SparkConf() \
                     .setAppName('task3') \
                     .setMaster('local[*]') \
                     .set('spark.driver.memory', '8g') \
                     .set('spark.executor.memory', '4g')

    sc = pyspark.SparkContext(conf = sc_conf)
    sc.setLogLevel('OFF')

    parser = argparse.ArgumentParser(description='A1T3')
    parser.add_argument('--input_file', type=str, default = './data/hw1/review.json', help ='input review file')
    parser.add_argument('--output_file', type=str, default = 'task3_default.json', help = 'outputfile')
    parser.add_argument('--n', type=int, default = '10', help = 'threshold number of reviews')
    args = parser.parse_args()

    '''
    YOUR CODE HERE
    '''
```

Figure 5. Skeleton code for task3 default.

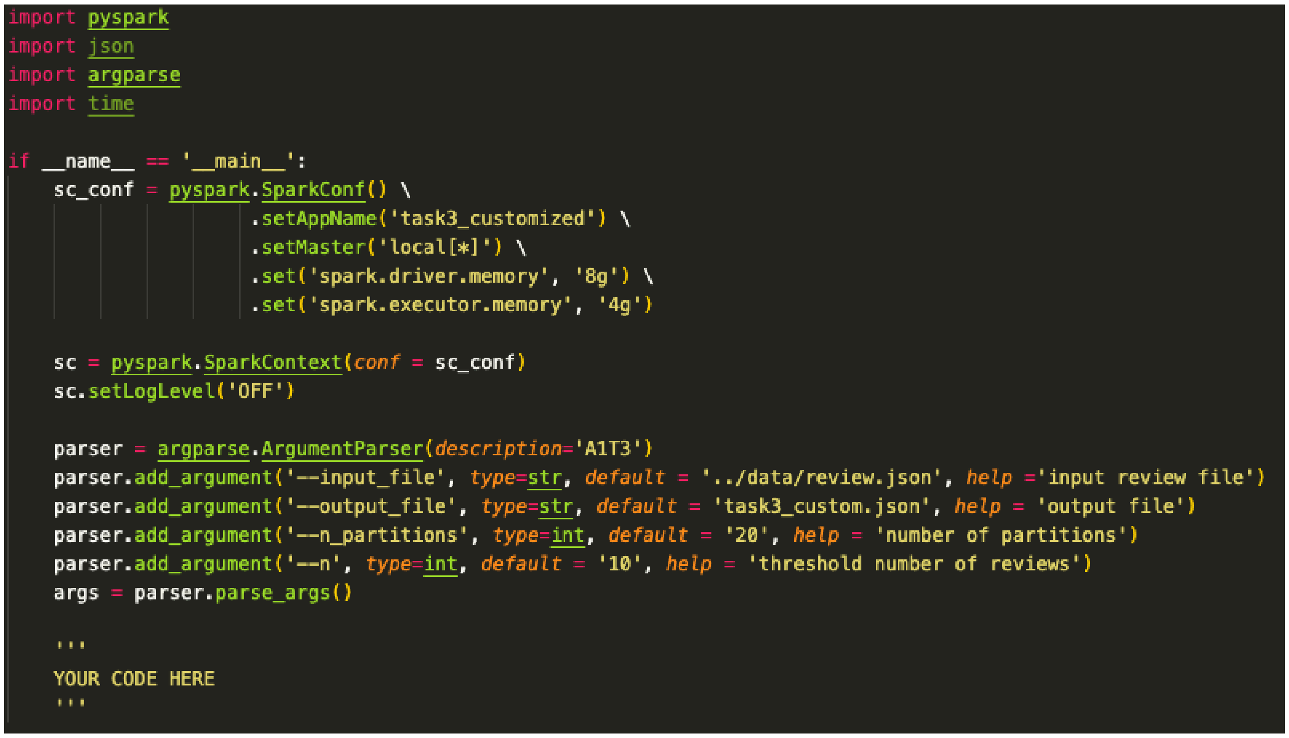
Python

```
python task3_default.py --input_file <input_file> --output_file
<output_file> --n <n>
```

Params:

- **`input_file`** – the input file (the review dataset)
- **`output_file`** – the output file contains your answers
- **n** – the threshold of the number of businesses (see 4.3.1)

We have provided the skeleton code pictured below in the **`task3_customized.py`** file. Please do not change the name of any of the command line parameters. Feel free to write your own helper functions. You may also import packages as long as they are default Python packages (see Section 2.1).



```python
import pyspark
import json
import argparse
import time

if __name__ == '__main__':
    sc_conf = pyspark.SparkConf() \
                    .setAppName('task3_customized') \
                    .setMaster('local[*]') \
                    .set('spark.driver.memory', '8g') \
                    .set('spark.executor.memory', '4g')

    sc = pyspark.SparkContext(conf = sc_conf)
    sc.setLogLevel('OFF')

    parser = argparse.ArgumentParser(description='A1T3')
    parser.add_argument('--input_file', type=str, default = '../data/review.json', help ='input review file')
    parser.add_argument('--output_file', type=str, default = 'task3_custom.json', help = 'output file')
    parser.add_argument('--n_partitions', type=int, default = '20', help = 'number of partitions')
    parser.add_argument('--n', type=int, default = '10', help = 'threshold number of reviews')
    args = parser.parse_args()

    '''
    YOUR CODE HERE
    '''
```

Figure 6. Skeleton code for task3 customized.

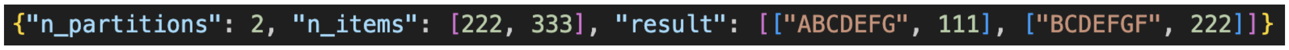
```
Python
python task3_customized.py --input_file <input_file> --output_file <output_file>
--n_partitions <n_partitions> --n <n>
```

Params:

input_file – the input file (the review dataset)
output_file – the output file contains your answers
n_partitions – the number of partitions (e.g., 10)
n – the threshold of the number of businesses (see 4.3.1)

4.3.3 Output format:

You must write the results in the JSON format using **exactly the same tags** (see an example in Figure 4). The answer for the number of partitions is a number. The answer for the number of items per partition is a list of numbers. The answer for the result is a list of pairs **[reviewers, count]** (no need to sort).



```
{"n_partitions": 2, "n_items": [222, 333], "result": [["ABCDEFG", 111], ["BCDEFGF", 222]]}
```

Figure 4: An example output for task3 in JSON format

6. Grading Criteria

(% penalty = % penalty of possible points you get)

1. You can use your free 5-day extension separately or together. During grading, TAs will count the number of late days **based on the submission time.** For example, if the due date is 2025/02/24 23:59:59 CDT and your submission is 2024/02/25 05:23:54 CDT, the number of late days would be 1. **TAs will record grace days by default (NO separate Emails)**. However, if you want to save it

for next time and treat your assignment as late submission (with some penalties), **please leave a comment in your submission.**

2. There will be no points given if your submission cannot be executed on the Gradescope.
3. There is no re-grading. Once the grade is posted on Canvas, we will only re-grade your assignments if there is a grading error. No exceptions.
4. Homework assignments are due at 11:59 pm CT on the due date and should be submitted on Canvas. Late submissions within 24 hours of the due date will receive a 30% penalty. Late submissions after 24 hours of the due date will receive a 70% penalty. Every student has FIVE free late days for homework assignments. You can use these free late days for any reason, separately or together, to avoid the late penalty. There will be no other extensions for any reason. **You cannot use the free late days after the last day of the class.**

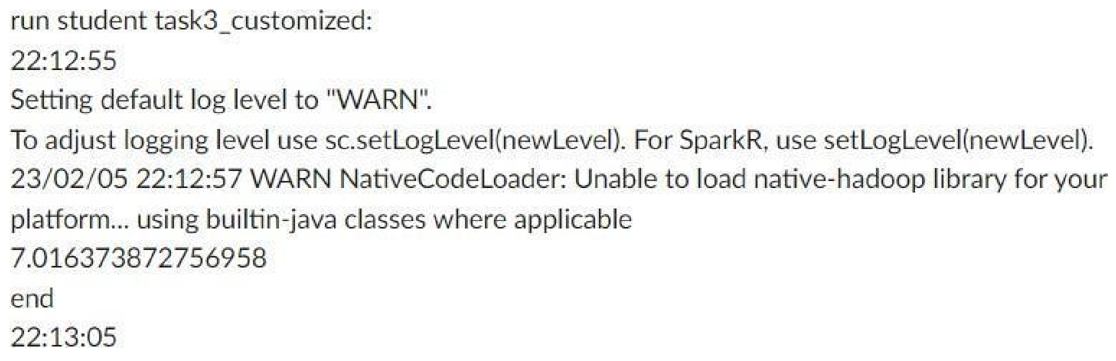
7. Submission Instructions

1. You will submit your programming assignment via Gradescope. You can access it from Canvas.
2. On Gradescope, you will find two submission links. One link is for the submission of **FOUR** Python files (**`.py`**), and the other link is for the submission of **ONE** PDF file for your answer for Task 3 (See Section 4.3.1).
3. For Python Codes: you need to submit 4 files (lowercase):
   `task1.py`
   `task2.py`
   `task3_default.py`
   `task3_customized.py`

4. For PDF file: you need to submit 1 file (lowercase):
   `task3.pdf`

5. Once you submit your programming files, the grading scripts on Gradescope will take some time (around 5-10 minutes) to run your files and run test cases. You should be able to see the final score for each task.

6. Besides the final score, you should be able to see the running time for task3 (both default and customized) in the following format (e.g., the task3_customized script takes about 7 seconds):



```
run student task3_customized:
22:12:55
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
23/02/05 22:12:57 WARN NativeCodeLoader: Unable to load native-hadoop library for your
platform... using builtin-java classes where applicable
7.016373872756958
end
22:13:05
```

7. You MUST follow the file naming provided in Step 3, 4 and 5. Failure to do this will result in failing to pass all test cases (even if your code is correct). It is your responsibility to make sure that your filenames strictly follow the instructions provided above.

8. If you did not pass a test case for a particular task, this means that there is an error that you have to fix. On the other hand, passing all test cases is a good sign, **but it does not mean that you will pass the test cases that the TA will use for grading**. Your final grade is ultimately determined by the test cases that are run after the due date (i.e., not the test cases that are available to students before the due date).

9. You are allowed to submit your codes to Gradescope as long as it is before the due date. If you submit your codes after the due dates, it will be recorded as a late submission (See Section 5 for information about grading criteria).