Background: Detecting popular and trending topics from news articles is important for public opinion monitoring. In this project, your task is to analyze text data over a dataset of Australian news from ABC (Australian Broadcasting Corporation) using Spark. The problem is to compute the yearly average term weights in the news articles dataset and find the top-k important terms.

Input file: The dataset you will use contains data from news headlines published over several years. In this text file, each line is a headline of a news article, in the format of "date,term1 term2 ... ... ". The date and text are separated by a comma, and the terms are separated by a space character. A sample file is like the one below:

20030219,council chief executive fails to secure position
20030219,council welcomes ambulance levy decision
20030219,council welcomes insurance breakthrough
20030219,fed opp to re intr
20040501,cowboys survive
20040501,cowboys withsta
20040502,castro vows cub
20200401,coronanomics th
20200401,coronavirus at h
20200401,coronavirus cam
20201016,china builds pig

When you click the panel o at has, in your home directory, a text file called " l free to open the file and explore its contents). The e
https://www.kaggle.com/th

Term weights computation: To compute the weight for a term regarding a year, please use the TF/IDF model. Specifically, the TF and IDF can be calculated as:

TF(term t, year y) = log10(the frequency of t in y)
IDF(term t, year y) = log10(the number of headlines in y/the number of headlines in y having t)

Please import math and use math.log10() to compute the term weights.

Finally, the term weight of term t in year y is computed as:
Weight(term t, year y) = TF(term t, year y) * IDF(term t, year y).

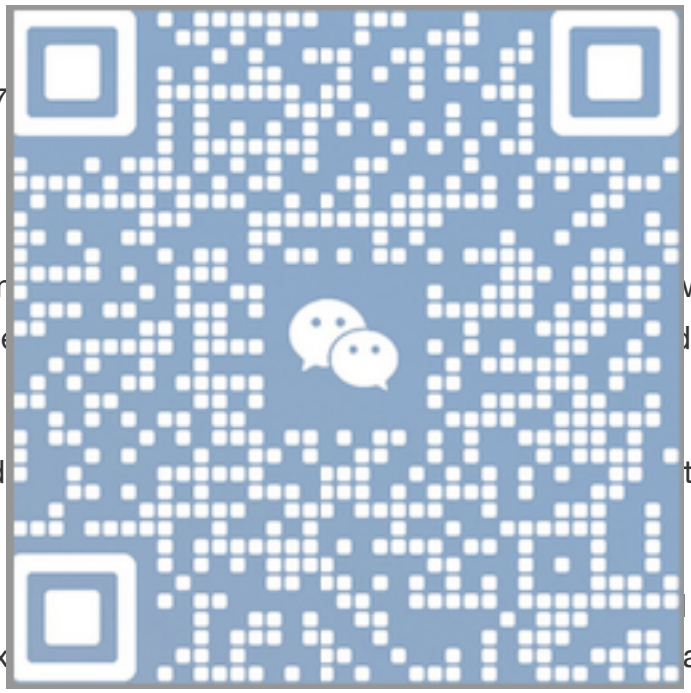
You may ignore small precision differences in the calculations.

Problem: Your task is to compute the term weights in each year, get the yearly average weight for each term, rank the results by the average weights in descending order first and then by the term alphabetically, and finally output the top-k terms and their yearly average weights.

It is also required to ignore the highly frequent terms (since they could be stopwords). You need to compute the global count for each term, rank them by their counts and break the ties by their alphabetical order, and the top-n terms should be filtered out from the dataset. In the above example, if n=1, you need to ignore "to".

You should output exactly k lines in your final output file. In each line, you need to output a pair of a term and the yearly average weight, in the format of "term\tWeight". For example, given the above data set, n=1, and k=5, the output should be:

"insurance" 0.090619058
"welcomes" 0.090619058
"coronavirus" 0.059610927
"council" 0.059610927
"cowboys" 0.053008751



Code Format: The code ter wo solutions, one using only RDD APIs and the othe take four parameters: the input file, the output folder,

$ spark-submit project2_rd tput" 1 5

Some notes
You can read the files from files is more convenient, but you must use the prefix ath does not have a prefix.

You are not allowed to use numpy or pandas, since we aim to assess your understanding of the RDD/DataFrame APIs.

You can use coalesce(1) to merge the data into a single partition and then save the data to disk.

In the DataFrame solution, it is not allowed to use the spark.sql() function to pass the SQL statement to Spark directly.

It does not matter if you have a new line at the end of the output file or not. It will not affect the correctness of your solution.

Marking Criteria (8 marks for each solution)
You must complete this assignment using Spark RDD/DataFrame APIs. Submissions only contain

regular Python techniques will be marked as 0.

Submission can be compiled and run on Spark => +3

Submission can obtain correct top-k results (including correct terms, weights, and order) => +3

Submissions correctly using Spark APIs (RDD/DataFrame solution only RDD/DataFrame APIs allowed) => +0.5

Submissions with excellent code format and structure, readability, and documentation => 0.5

Submissions efficiently removing highly frequent terms => 1