(https://)# ASSIGNMENT 2 - UE22CS343AB2 - BIG DATA

Introduction

This document contains 2 tasks :
Task 1 : This task is to be solved using Spark
Task 2 : This task is to be solved using Kafka

Objectives

- Get familiar with Kafka
- Get familiar with Spark

Ethical Practices

Please submit original code only. All solutions must be submitted through the portal. We will perform a plagiarism check on the code and you will be penalized if your code is found to be plagiarized.

Submission Deadline

Thursday, 24th October, 2024.

The portal uptimes will be communicated to you in due course. Please do not wait till the last minute to submit your code since wait times may be long.

Submission Link

Assignments need to be submitted on portal: https://www.bigdata-pesu-2024.tech (https://www.bigdata-pesu-2024.tech)
(the site is offline right now, the online timings will be communicated via groups and mails).

Software/Language to be used

1. *Python 3.10.x*
2. Apache Spark v3.5.3
3. Apache Kafka v3.7.1

Additionally, the following Python libraries are required:

pyspark == 3.5.3

kafka-python == 2.0.2

No other libraries are allowed.

Submission Guidelines

You will need to make the following changes to your Python scripts before submitting them.

Include the following shebang at the top of your Python scripts.

```
#!/usr/bin/env python3
```

Convert your files to executable using the following command

```
chmod +x *.py
```

Convert line breaks in DOS format to Unix format (this is necessary if you are coding on Windows without which your code will not run on our portal).

```
dos2unix *.py
```

Task 1 - Spark

Problem Statement

The International Olympic Committee is preparing for their centennial celebration and wants to recognize outstanding athletes and coaches from the past decade. They've asked you, as a data analyst, to help them identify the top performers based on the data from 2012, 2016, and 2020 Olympics.

Dataset

Dataset to test your code (https://drive.google.com/drive/folders/1YgY3UCZvLiuc6A-1qQCs4V1DY9tqNNHV?usp=sharing)

You are provided with 2 folders in the above link.

1. Small Dataset : This dataset can be used to understand how the points tallying works and users should approach the problem
2. Large Dataset : This will simulate the large dataset you will be provided on the portal. Test your code with this dataset to check how its performing with space and

time complexity.

> **However, keep in mind that the final dataset on the portal is larger than what is given to you and there is no guarantee that if you pass the local large dataset, you will pass on the portal. Understading the problem and the constraints is key to get the right solution.**

Dataset Description

1. **You are provided with 3 datasets for the athletes for each year 2012,2016,2020 that contains the following columns :**

Column	Description
id	A unique ID for each athlete
name	The name of the athlete
dob	Date of Birth
height(m)	Height in meters
weight(kg)	Weight in kilograms
sport	The sport the athlete participated in
event	The event the athlete participated in his respective sport
country	The nationality of the athlete
num_followers	Number of followers the athlete has
num_articles	Number of articles published under the athlete names
personal_best	The athlete's best performance for the sport on/before that respective year
coach_id	The athlete's coach for that year

2. **You are provided with 1 dataset for the coaches over the years 2012,2016,2020 that contains the following columns:**

Column	Description
id	The unique ID for the coach
name	The name of the coach
age	Age of the coach
years_of_experience	How many years of experience the coach has
sport	The sport the coach trains athletes for
country	The nationality of the coach
contract_id	The unique ID of the coach's contract for the tenure of his coaching
certification_committee	The committee under which the coach obtained certification

3. **Finally, one dataset to cover the medals obtained by the athletes over the years 2012,2016,2020. It contains the following columns:**

Column	Description
id	The unique ID for the athlete who won the medal
sport	Sport under which the medal was won
event	The event under which the medal was obtained
year	The year of winning the medal
country	The country that hosted the Olympics for that year
medal	The category of the medal won (Gold/Silver/Bronze)

Task 1.1: Identifying the All-Time Best Athletes

Description:

The IOC wants to honor the best athlete in each sport across all three Olympic years (2012, 2016, and 2020). Your job is to analyze the data and determine which athlete has performed the best in their respective sport over this period. Consider the following table to measure the total points for each athlete over all three olympics and rank them on that metric.

Scoring Metrics:

2012 :

Medal Category	Points
Gold	20
Silver	15
Bronze	10

> **What the above table describes:**
> 1 gold medal is worth 20 points in 2012. Similarly in 2012, 1 silver is worth 15 points and 1 bronze is worth 10 points.

2016 :

Medal Category	Points
Gold	12
Silver	8
Bronze	6

2020 :

Medal Category	Points
Gold	15
Silver	12
Bronze	7

> **NOTE : Best performance is considered as the most points won.**

Task 1.2: Recognizing Top International Coaches

Description:

To promote international cooperation, the IOC wants to acknowledge coaches who have successfully trained athletes from different countries. They've asked you to focus on three specific countries: **China**, **India**, **USA** and identify the **top 5** coaches who have trained athletes from these nations. Your analysis should consider and rank coaches by aggregating the total points won by the athletes under them. For this, you may refer to the same scoring metric given above.

> **The OUTPUT should all be in uppercase.**

Points to keep in mind while solving the Task

Here are some points to keep in mind about the datasets provided :

1. A coach teaches only 1 country per year. He cannot coach multiple countries simultaneously but keep in mind, he can change countries over the years i.e. coach country X in 2012 and a different country Y in 2016.
2. The sport of the athlete must match the sport of the coach.
3. The data provided you can be dirty and have years we don't want in the final answer. The only years to be considered for the analysis are 2012,2016,2020. All other years are null and void.
4. The dataset has random casing. Fix the randomness by converting all to uppercase.
5. In case of a tie on total points, the athlete/coach with more gold medals comes first. If number of golds are same, consider silver and so on. If two or more coaches have the same number of points, gold, silver and bronze medals then pick the athlete/coach with the lexicographic **lowest** name.

> **NOTE : The lowest lexicographic name is the first name when all the names are arranged alphabetically or are in dictionary order.**

Working with the files

Working with CSV Files in PySpark :

1. PySpark's `read.csv()` is the preferred method to read the files.
   An example to import the csv is as follows :
   ```
   athlete_2012 = spark.read.csv(sys.argv[1], header=True, inferSchema=True)
   ```
   Explaination :
   ~~~
   ```
   header=True: This ensures that the first row of the CSV is treated as column header
   inferSchema=True: This automatically infers the data type of each column.
   ```
   ~~~

2. After loading the datasets, you can perform various transformations, filters, or joins to process the data as needed.

3. If you wish to see the whole csv, you can store it using `write.csv(output_path, header=True, mode="__")`

4. The expected output however requires only the name, so extract only that column and write it to a txt file. The text file is expected to contain the names seperated with ',' in the following format and **ensure there are no extra whitespaces**

5. The command to execute the python file should take this structure:
 `python task1.py athletes_2012.csv athletes_2016.csv athletes_2020.csv coaches.csv medals.csv output.txt`
 where, output.txt will hold you final output with the structure shown above

> **Students are expected to submit one input code file which outputs one .txt file with the answer to both tasks**

> **We expect you to output your final answer into the txt file in the following format** `([task1.1 comma seperated values],[task1.2 comma seperated values])`
> Example:
> `([athlete A,athlete B,....],[coach A,coach B,...])`
>
> When writing `[task1.1 comma seperated values]` write the athlete names sorted according to **sport alphabetically**
>
> Similarly,
> When writing `[task1.2 comma seperated values]` write the top 5 coaches of **China first, then India, then USA**

Task 2 - Kafka

Story Background

You are a developer for an online coding platform which hosts various DSA problems and competitions for students, developers and anyone needing interview preparations. There is continuous stream of information generated by the platform and you are tasked with building a producer-consumer pipeline with Kafka to support the various operations and computations needed for the platform.

Dataset Format and Explanation

The rows in the dataset are of 3 types, based on the event that they are describing -

1. `problem` : Describes the event when a user has made a submission for a specific problem with additional fields. The schema for this format is -

   > problem user_id problem_id category difficulty submission_id status language runtime

An example row of this format:

`problem u_1 p_1 Arrays Medium s_001 Passed Python 300`

2. `competition` : Describes the event when a user has made a submission for a problem as a part of the competition with additional fields. The schema for this format is -

   > competition comp_id user_id com_problem_id category difficulty comp_submission_id status language runtime time_taken

An example row of this format:

`competition c_1 u_1 cp_1 Trees Hard cs_1 Failed C 50 22`

3. `solution` : Describes the event when a user has made his correct/passed submission public for other users to see. The schema for this format is -

   > solution user_id problem_id submission_id upvotes

An example row of this format:

`solution u_1 p_1 s_1 230`

Additional Details about the dataset

1. The rows will be shuffled, as it is a stream of events.
2. The unit of measurement for runtime is millisecond(ms) and for time_taken is minutes.
3. The problems and solutions in the `competition` format are independent/different from those in the `problem` format.
4. If a submission is present in the `solution` format, it will also be present in the `problem` format with status `Passed` .

5. In the `competition` format, the same `comp_problem_id` can be used for multiple competitions.

Problem Statement

Using the dataset provided to you, generate output files for 3 different clients based on their requirements.
Client 1: Wants to know the most frequently used programming language and the most difficult category to solve.
Client 2: Wants to get the leaderboard for all competitions with the points each user scored in the each competition.
Client 3: Wants to know which user is contributing to the community the most (highest total upvotes), and additionally also wants to calculate every user's elo rating on the platform.

Description

Sample dataset

```
competition c_0003 u_012 cp_00004 Stacks&Queues Hard cs_00003 Failed Rust 1994 46
problem u_019 p_00006 DP Hard s_00005 Passed C 1408
problem u_016 p_00008 Trees Hard s_00003 Failed Java 1443
problem u_009 p_00015 Graphs Hard s_00004 Passed Go 120
solution u_019 p_00006 s_00005 511
problem u_017 p_00009 Heaps Hard s_00002 Failed Go 821
solution u_009 p_00015 s_00004 180
competition c_0003 u_018 cp_00008 Heaps Easy cs_00005 Passed Go 411 7
competition c_0001 u_005 cp_00004 Stacks&Queues Hard cs_00004 Passed Ruby 756 25
competition c_0002 u_002 cp_00010 Greedy Hard cs_00001 TLE Python 2440 32
competition c_0002 u_003 cp_00010 Greedy Hard cs_00002 TLE Python 2076 18
problem u_020 p_00016 Stacks&Queues Easy s_00001 TLE C++ 2805
EOF
```

Client 1

Needs information on 2 things -

- Most frequently used programming language, i.e, the language with the most submissions made
- Most difficult category to solve, i.e, the category with the smallest ratio of `Passed/Total` submissions

Sample output and format after running consumer1.py (http://consumer1.py), for client1: Print the json.dumps of the dictionary in the following format -

```
{
    "most_used_language": [
        "Go"
    ],
    "most_difficult_category": [
        "Heaps",
        "Stacks&Queues",
        "Trees"
    ]
}
```

Notes for client 1:

1. The client is only interested in the submissions made in the `problem` format.
2. If there is a tie for either part of the question, then return all the results in a list sorted lexicographically(similar to the above output where Heaps and Trees are tied)

Client 2

Needs leaderboard with points for all competitions. You need to compute the points scored in each competition by a user to find the final leaderboard.

Sample output and format after running consumer2.py (http://consumer2.py), for client2: Print the json.dumps of the dictionary in the following format -

```
{
    "c_0001": {
        "u_005": 2393
    },
    "c_0002": {
        "u_002": 60,
        "u_003": 79
    },
    "c_0003": {
        "u_012": 0,
        "u_018": 2358
    }
}
```

Notes for client 2:

1. Needs the leaderboard in lexicographically sorted order of competition_id.
2. Under each competition, there must be key-value pairs of `user:points_scored` for all the users in the competition.
3. The users in each competition must be sorted in lexicographic order of user_id.
4. `Round down` the final points scored to an integer in each competition for each user.

5. Formula for points for each submission is given below 👇

```
Submission_Points = Status_Score * Difficulty_Score * Bonus

Status_Score = 100 if Passed | 20 if TLE | 0 if Failed
Difficulty_Score = 3 if Hard | 2 if Medium | 1 if Easy
Bonus = max(1,(1 + Runtime_bonus - Time_taken_penalty))
Runtime_bonus = 10000/runtime
Time_taken_penalty = 0.25*time_spent
```

Client 3

Needs information on 2 things -

- User with the most contribution to the community, i.e, the user with the most total upvotes.
- Each user's elo rating.

Sample output and format after running consumer3.py (http://consumer3.py), for client3:
Print the json.dumps of the dictionary in the following format -

```
{
    "best_contributor": [
        "u_019"
    ],
    "user_elo_rating": {
        "u_002": 1210,
        "u_003": 1211,
        "u_005": 1245,
        "u_009": 1315,
        "u_012": 1195,
        "u_016": 1197,
        "u_017": 1202,
        "u_018": 1233,
        "u_019": 1239,
        "u_020": 1205
    }
}
```

Notes for client 3:

1. If the best_contributor is tied on upvotes, then list all the user_id's sorted lexicographically.
2. The users with elo ratings are sorted by user_id lexicographically.
3. `Round down` the user elo rating to an integer
4. Both `competition` and `problem` submissions are considered for the elo rating of a user.

5. Formula for User Elo Rating -

```
New_Elo = Current_Elo + Submission_Points

Submission_Points = K * (Status_Score * Difficulty_Score) + Runtime_bonus
K = 32 (constant scaling factor)
Status_Score = 1 if Passed, 0.2 if TLE, -0.3 if Failed
Difficulty_Score = 1 if Hard, 0.7 if Medium, 0.3 if Easy
Runtime_bonus = 10000/runtime


Initial elo rating for all users = 1200
```

Tips to solve the Task

The logic for the producer and consumer is up to you. The input to the producer file will be streamed through the standard input.

You are required to use the `kafka-python` library to solve the problem statement. You should have four files, one for the producer and three for the three different consumers. The producer should be named `kafka-producer.py` and the consumers should be named `kafka-consumer1.py`, `kafka-consumer2.py` and `kafka-consumer3.py`.

It is recommended for you to use three topics to solve the assignment. All three topic names will be passed as command line arguments to all four files, and you may make use of them as required. There is no constraint on how you want to use these 3 topics.

To test your code, run the consumer files first in three seperate terminals and then the producer file in a fourth separate terminal.

```
./kafka-consumer1.py topicName1 topicName2 topicName3 > output1.json
```

```
./kafka-consumer2.py topicName1 topicName2 topicName3 > output2.json
```

```
./kafka-consumer3.py topicName1 topicName2 topicName3 > output3.json
```

```
cat sample_dataset.txt | ./kafka-producer.py topicName1 topicName2 topicName3
```

Important

1. The topic name should not be hardcoded. Three topic names should be passed as a command line argument for both the producer and consumer files.
2. There is a special line in the end of the input file named `EOF`. This is done so that the consumer knows when to stop reading from the topic. You must not include this line in the output.

3. Usage of direct file interaction commands such as python's `open()` is not allowed. You must use Kafka's producer and consumer APIs to solve the problem statement.
4. Only `kafka` , `sys` , `json` and `math` modules are allowed.
5. Print the `result` dictionary in each consumer file using the following command only
   `print(json.dumps(result, indent = 4))`

Dataset

You can find the datasets and expected outputs here (https://drive.google.com/drive/folders/1mV6DqNsjo91zGMQtgxpD8q0goy4PFZ__?usp=sharing)