Assignment 2
Report for TUT07 F10A-10
SID 500046368, 500117613
Group member: Dawei Yun, Jinpeng Han

Question1

1. SQL code preprocessing

There are several inconsistencies and syntax errors in the provided SQL code, such as 'United States' and 'USA', 'BOEING' and 'Boeing'.

After modifying them, the following results can be obtained.

Table +

	name	count(tail_number)
1	Southwest Airlines Co.	11912
2	Delta Air Lines Inc.	5829
3	United Airlines	5248
4	Continental Air Lines Inc.	3421
5	AirTran	2184

5 rows | 13.05 seconds runtime

2. Rewrite as Apache Spark Python code (pySpark)

When rewrite the SQL query into PySpark, first I read all the datasets in the format of dataframes. For the 'flight_medium' and 'flight_large' datasets, which have not been cleaned, I remove all null values and then corrected the column names.

3. Explanation of pySpark code

In the PySpark code, first I join the three datasets that are needed, using an inner join between the medium dataset and the large dataset. Then filter the merged dataset based on the conditions where country == 'United States' and manufacturer == 'BOEING'.

After that I perform a groupby operation on the filtered dataset. Using count, and sort it in descending order based on the 'count_tail_number ' column. Finally, display only the first five rows of the resulting dataset.

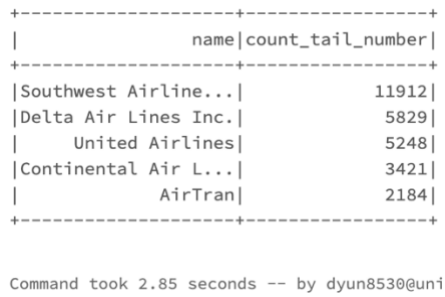
4. Optimization

In terms of optimizing code performance, I have found that filtering the datasets first and

then performing the join can significantly reduce the amount of data that needs to be processed during the join operation. Therefore, improving the runtime. From the results, it is indeed very effective.

Inefficient code

Run time and result for small dataset.

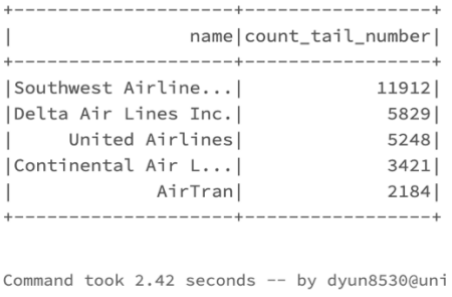


```
+--------------------+-----------------+
|                name|count_tail_number|
+--------------------+-----------------+
|Southwest Airline...|            11912|
|Delta Air Lines Inc.|             5829|
|     United Airlines|             5248|
|Continental Air L...|             3421|
|             AirTran|             2184|
+--------------------+-----------------+

Command took 2.85 seconds -- by dyun8530@uni.
```

Efficient code

Run time and result for small dataset.

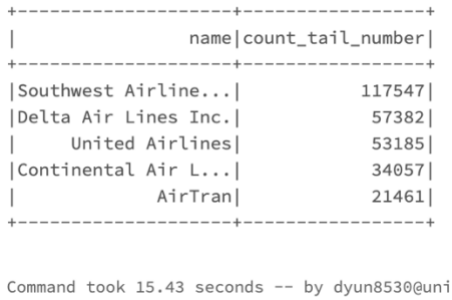


```
+--------------------+-----------------+
|                name|count_tail_number|
+--------------------+-----------------+
|Southwest Airline...|            11912|
|Delta Air Lines Inc.|             5829|
|     United Airlines|             5248|
|Continental Air L...|             3421|
|             AirTran|             2184|
+--------------------+-----------------+

Command took 2.42 seconds -- by dyun8530@uni
```

Inefficient code

Run time and result for medium dataset.

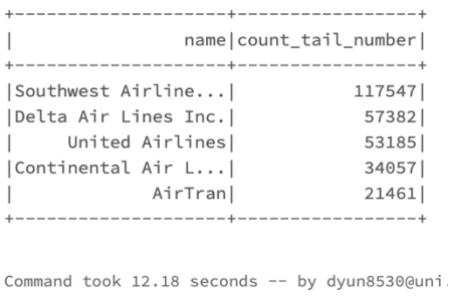


```
+--------------------+-----------------+
|                name|count_tail_number|
+--------------------+-----------------+
|Southwest Airline...|           117547|
|Delta Air Lines Inc.|            57382|
|     United Airlines|            53185|
|Continental Air L...|            34057|
|             AirTran|            21461|
+--------------------+-----------------+

Command took 15.43 seconds -- by dyun8530@uni
```

Efficient code

Run time and result for medium dataset.

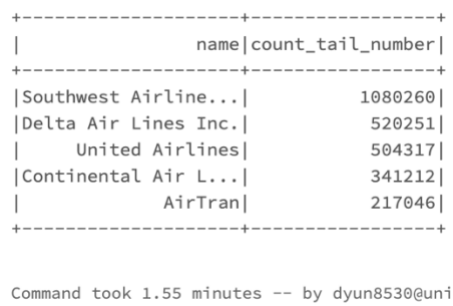


```
+--------------------+-----------------+
|                name|count_tail_number|
+--------------------+-----------------+
|Southwest Airline...|           117547|
|Delta Air Lines Inc.|            57382|
|     United Airlines|            53185|
|Continental Air L...|            34057|
|             AirTran|            21461|
+--------------------+-----------------+

Command took 12.18 seconds -- by dyun8530@uni
```

Inefficient code

Run time and result for large dataset.

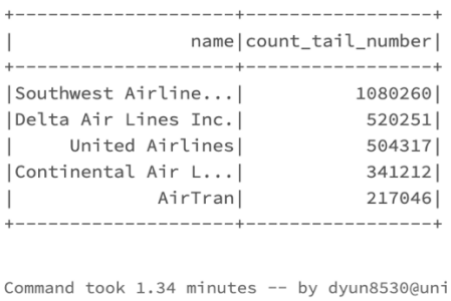


```
+--------------------+-----------------+
|                name|count_tail_number|
+--------------------+-----------------+
|Southwest Airline...|          1080260|
|Delta Air Lines Inc.|           520251|
|     United Airlines|           504317|
|Continental Air L...|           341212|
|             AirTran|           217046|
+--------------------+-----------------+

Command took 1.55 minutes -- by dyun8530@uni
```

Efficient code

Run time and result for large dataset.



```
+--------------------+-----------------+
|                name|count_tail_number|
+--------------------+-----------------+
|Southwest Airline...|          1080260|
|Delta Air Lines Inc.|           520251|
|     United Airlines|           504317|
|Continental Air L...|           341212|
|             AirTran|           217046|
+--------------------+-----------------+

Command took 1.34 minutes -- by dyun8530@uni
```

5. Comparison in Stagemetrics

For query plan analysis, I compared the efficiency using the 'stagemetrics' module from the 'sparkmeasure' package.

Inefficient code runtime for small dataset.

```
elapsedTime => 2010 (2 s)
stageDuration => 2047 (2 s)
executorRunTime => 3956 (4 s)
executorCpuTime => 1121 (1 s)
```

Efficient code runtime for small dataset.

```
elapsedTime => 1863 (2 s)
stageDuration => 1817 (2 s)
executorRunTime => 3712 (4 s)
executorCpuTime => 928 (0.9 s)
```

Inefficient code runtime for medium dataset.

```
elapsedTime => 11810 (12 s)
stageDuration => 12009 (12 s)
executorRunTime => 76882 (1.3 min)
executorCpuTime => 13019 (13 s)
```

Efficient code runtime for medium dataset.

```
elapsedTime => 10930 (11 s)
stageDuration => 10990 (11 s)
executorRunTime => 74141 (1.2 min)
executorCpuTime => 12712 (13 s)
```

Inefficient code runtime for large dataset.

```
elapsedTime => 83920 (1.4 min)
stageDuration => 196230 (3.3 min)
executorRunTime => 621775 (10 min)
executorCpuTime => 117153 (2.0 min)
```

Efficient code runtime for large dataset.

```
elapsedTime => 83455 (1.4 min)
stageDuration => 137874 (2.3 min)
executorRunTime => 606799 (10 min)
executorCpuTime => 116106 (1.9 min)
```

Here, efficient code is superior in all aspects to inefficient code.

6. Comparison in SQL query plan

When comparing the SQL execution plans, I observed that these two execution plans are very similar. However, I noticed that the execution plan of the less efficient code includes an "AQEShuffleRead coalesced" operation, while its not in efficient code.

According to the Databricks website, Adaptive Query Execution (AQE) is a feature that automatically optimizes the physical execution strategy. This mean that when spark executing the code less efficient, it automatically optimized the execution using AQE. This further indicates the efficiency difference between the two codes, where the more efficient code may have already achieved a high level of efficiency and hence did not require optimization through AQE.

7. Comparison in DAG visualization

In terms of DAG visualization, the workflow of the two code implementations is nearly the same. The only difference is which step they did the AQEShuffleRead operation. The location of AQEShuffleRead is not fixed and can change depending on when you run the code. Therefore, we can't use the AQEShuffleRead as a measurement of efficiency.

Additionally, there is a slight difference in the time taken for the third CSV file reading in the DAG visualization between the two codes. The less efficient code takes approximately 0.1 minutes longer to read the third CSV file compared to the more efficient code. This further explain the importance of doing the filter operations early, because join and groupby operations will benefit from the reduced dataset size.

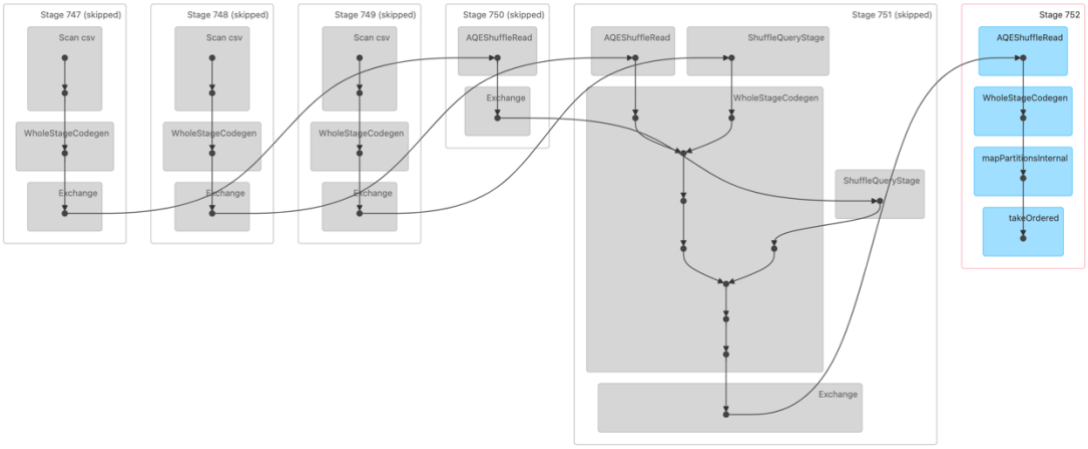
The third csv read time for inefficient code.

from sparkmeasure import StageMetrics stagemetr... $anonfun$withThreadLocalCaptured$1 at CompletableFuture.java:1604	2023/05/28 08:34:54	1.4 min

The third csv read time for efficient code.

from sparkmeasure import StageMetrics stagemetr... ')$anonfun$withThreadLocalCaptured$1 at CompletableFuture.java:1604	2023/05/28 08:32:44	1.3 min

The overall DAG Visualization, it is not fixed and similar for two codes.



Question 2

1. **Data preprocessing**

   After loading the data, the first task is to standardize the column names of the medium sized dataset and the large dataset. Next, I changed the time format. My method is splitting the four-digit time into two separate columns, representing hours and minutes. This enables efficient calculations, where I convert the hour data to minutes by multiplying it by 60 to obtain the delay time. Lastly, I only keep the required columns, minimizing the computational time.

2. **Explanation of pySpark code**

   To begin with, I apply data filtering on the dataset based on the user-specified year and country. Using the flight dataset, I perform a groupby operation to compute the total delay time for each airport. Then, I proceed with join and orderby functions to identify the top five airports with the highest delay times.

   Then, I perform a join operation between the airline dataset and the flight dataset. Using groupby and orderby functions, I obtain the delay time for each airline corresponding to the departure airport.

   Finally, I join the two resultant datasets and conduct calculations and ranking. This allows me to determine the delay time for each airport and identify the top five airlines with the longest delays. I also include calculate the percentage of total delay time attributed to these airlines.

3. **Optimization**

   In terms of optimization, First, I filter the data based on the user-specified country and year before I start preprocessing the data. This significantly reduces the data volume, resulting in reduced running time for processing time formatting and calculating delay time.

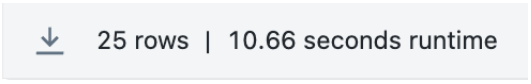
   Next, I use broadcast join instead of join operations. Broadcast join can avoid reading repetitive data, thus improving efficiency. Additionally, I use the concept of caching, it is slightly different from the description said. I create a cache for data frames that are used multiple times, instead of automatically caching after each run. In fact, I include a code to clear the cache. Caching helps avoid superfluous data reads and calculations, therefore enhancing the execution efficiency of the code.

4. Result and running time.

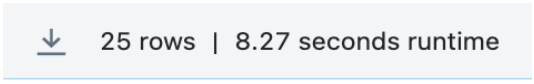
Result for small dataset. (Both Inefficient code and efficient code)

	airport_name	total_delay_minute	name	percentage
1	Chicago O'Hare International	3894	United Airlines	59.27
2	Chicago O'Hare International	3894	American Airlines Inc.	24.5
3	Chicago O'Hare International	3894	Northwest Airlines Inc.	8.04
4	Chicago O'Hare International	3894	Continental Air Lines Inc.	2.82
5	Chicago O'Hare International	3894	US Airways	2.72
6	Denver Intl	2695	United Airlines	96.1
7	Denver Intl	2695	American Airlines Inc.	1.52
8	Denver Intl	2695	US Airways	1.45
9	Denver Intl	2695	Delta Air Lines Inc.	0.45
10	Denver Intl	2695	Tway Air	0.19
11	Los Angeles International	2684	United Airlines	40.54
12	Los Angeles International	2684	Southwest Airlines Co.	23.17
13	Los Angeles International	2684	American Airlines Inc.	17.7
14	Los Angeles International	2684	Delta Air Lines Inc.	7.27
15	Los Angeles International	2684	Tway Air	3.06
16	Miami International	2634	American Airlines Inc.	73.08
17	Miami International	2634	United Airlines	25.78
18	Miami International	2634	Tway Air	0.99
19	Miami International	2634	US Airways	0.11
20	Miami International	2634	Delta Air Lines Inc.	0.04
21	William B Hartsfield-Atlanta Intl	3417	Delta Air Lines Inc.	76.27
22	William B Hartsfield-Atlanta Intl	3417	Sparrow Aviation	12.06
23	William B Hartsfield-Atlanta Intl	3417	Northwest Airlines Inc.	3.37
24	William B Hartsfield-Atlanta Intl	3417	US Airways	2.52
25	William B Hartsfield-Atlanta Intl	3417	Continental Air Lines Inc.	2.49

Running time for Inefficient code.



Running time for efficient code.



Result for medium dataset. (Both Inefficient code and efficient code)

	airport_name	total_delay_minute	name	percentage
1	Chicago O'Hare International	44034	United Airlines	57.51
2	Chicago O'Hare International	44034	American Airlines Inc.	34.18
3	Chicago O'Hare International	44034	US Airways	2.96
4	Chicago O'Hare International	44034	Northwest Airlines Inc.	1.71
5	Chicago O'Hare International	44034	Continental Air Lines Inc.	1.26
6	Dallas-Fort Worth International	25750	American Airlines Inc.	72.29
7	Dallas-Fort Worth International	25750	Delta Air Lines Inc.	13.43
8	Dallas-Fort Worth International	25750	United Airlines	7.61
9	Dallas-Fort Worth International	25750	Continental Air Lines Inc.	2.1
10	Dallas-Fort Worth International	25750	Northwest Airlines Inc.	1.84
11	Los Angeles International	28732	United Airlines	45.66
12	Los Angeles International	28732	Southwest Airlines Co.	20.02
13	Los Angeles International	28732	American Airlines Inc.	13.5
14	Los Angeles International	28732	Delta Air Lines Inc.	4.71
15	Los Angeles International	28732	Alaska Airlines Inc.	3.14
16	Phoenix Sky Harbor International	26978	Sparrow Aviation	48.91
17	Phoenix Sky Harbor International	26978	Southwest Airlines Co.	29.3
18	Phoenix Sky Harbor International	26978	Delta Air Lines Inc.	7.57
19	Phoenix Sky Harbor International	26978	United Airlines	7.02
20	Phoenix Sky Harbor International	26978	Continental Air Lines Inc.	2.88
21	William B Hartsfield-Atlanta Intl	23820	Delta Air Lines Inc.	81.94
22	William B Hartsfield-Atlanta Intl	23820	United Airlines	5.79
23	William B Hartsfield-Atlanta Intl	23820	US Airways	4.34
24	William B Hartsfield-Atlanta Intl	23820	Northwest Airlines Inc.	3.06
25	William B Hartsfield-Atlanta Intl	23820	Continental Air Lines Inc.	1.8

Running time for Inefficient code.

Running time for efficient code.

25 rows | 56.28 seconds runtime

25 rows | 26.24 seconds runtime

Result for Large dataset. (Both Inefficient code and efficient code)

	airport_name	total_delay_minute	name	percentage
1	Chicago O'Hare International	450445	United Airlines	57.03
2	Chicago O'Hare International	450445	American Airlines Inc.	33.22
3	Chicago O'Hare International	450445	US Airways	2.38
4	Chicago O'Hare International	450445	Delta Air Lines Inc.	2.21
5	Chicago O'Hare International	450445	Continental Air Lines Inc.	1.72
6	Dallas-Fort Worth International	259775	American Airlines Inc.	70.06
7	Dallas-Fort Worth International	259775	Delta Air Lines Inc.	16.36
8	Dallas-Fort Worth International	259775	United Airlines	7.36
9	Dallas-Fort Worth International	259775	Continental Air Lines Inc.	2.24
10	Dallas-Fort Worth International	259775	Northwest Airlines Inc.	1.42
11	Los Angeles International	285210	United Airlines	41.71
12	Los Angeles International	285210	Southwest Airlines Co.	21.92
13	Los Angeles International	285210	American Airlines Inc.	13.18
14	Los Angeles International	285210	Delta Air Lines Inc.	7.76
15	Los Angeles International	285210	Sparrow Aviation	4.1
16	Phoenix Sky Harbor International	274688	Sparrow Aviation	43.23
17	Phoenix Sky Harbor International	274688	Southwest Airlines Co.	30.37
18	Phoenix Sky Harbor International	274688	Delta Air Lines Inc.	9
19	Phoenix Sky Harbor International	274688	United Airlines	8.21
20	Phoenix Sky Harbor International	274688	American Airlines Inc.	2.35
21	William B Hartsfield-Atlanta Intl	270080	Delta Air Lines Inc.	82
22	William B Hartsfield-Atlanta Intl	270080	American Airlines Inc.	4.21
23	William B Hartsfield-Atlanta Intl	270080	United Airlines	3.86
24	William B Hartsfield-Atlanta Intl	270080	US Airways	3.64
25	William B Hartsfield-Atlanta Intl	270080	Continental Air Lines Inc.	3.12

Running time for Inefficient code.

25 rows | 6.42 minutes runtime

Running time for efficient code.

25 rows | 3.86 minutes runtime

5. Comparison in Stagemetrics

For query plan analysis, I compared the efficiency using the 'stagemetrics' module from the 'sparkmeasure' package.

Inefficient code runtime for small dataset.

```
elapsedTime => 11539 (12 s)
stageDuration => 15881 (16 s)
executorRunTime => 36876 (37 s)
executorCpuTime => 9040 (9 s)
```

Efficient code runtime for small dataset.

```
elapsedTime => 7178 (7 s)
stageDuration => 6924 (7 s)
executorRunTime => 14655 (15 s)
executorCpuTime => 3285 (3 s)
```

Inefficient code runtime for medium dataset.

```
elapsedTime => 55515 (56 s)
stageDuration => 78695 (1.3 min)
executorRunTime => 378762 (6.3 min)
executorCpuTime => 71943 (1.2 min)
```

Efficient code runtime for medium dataset.

```
elapsedTime => 26391 (26 s)
stageDuration => 36264 (36 s)
executorRunTime => 156956 (2.6 min)
executorCpuTime => 26685 (27 s)
```

Inefficient code runtime for large dataset.

```
elapsedTime => 383996 (6.4 min)
stageDuration => 549974 (9.2 min)
executorRunTime => 2778037 (46 min)
executorCpuTime => 572319 (9.5 min)
```

Efficient code runtime for large dataset.

```
elapsedTime => 236926 (3.9 min)
stageDuration => 391223 (6.5 min)
executorRunTime => 1781239 (30 min)
executorCpuTime => 344730 (5.7 min)
```

Here, efficient code is superior in all aspects to inefficient code.

6. Comparison in SQL query plan

In terms of the physical plan, I observed that efficient code includes the statement.

{'PartitionFilters: [], PushedFilters: [IsNotNull(country), EqualTo(country, USA)].'}

This filters statement is located near the beginning and is absent in inefficient code. Applying filters early can result in decreased running time.

Additionally, inefficient code contains the 'HashAggregate' operation, whereas efficient code uses 'TakeOrderedAndProject.' For large datasets like 'flight_large,' HashAggregate will cause data overflow and result in additional running time.

7. Comparison in DAG visualization

"In terms of DAG Visualization, there is a significant difference in performance between two codes when processing Scan CSV to Exchange. In the efficient code, each ScanCSV process includes the operation of InMemoryTableScan. This is because I use caching operations in the efficient code, which speeds up the processing of accessed data. The largest time difference between the two code variations also arises from this aspect."

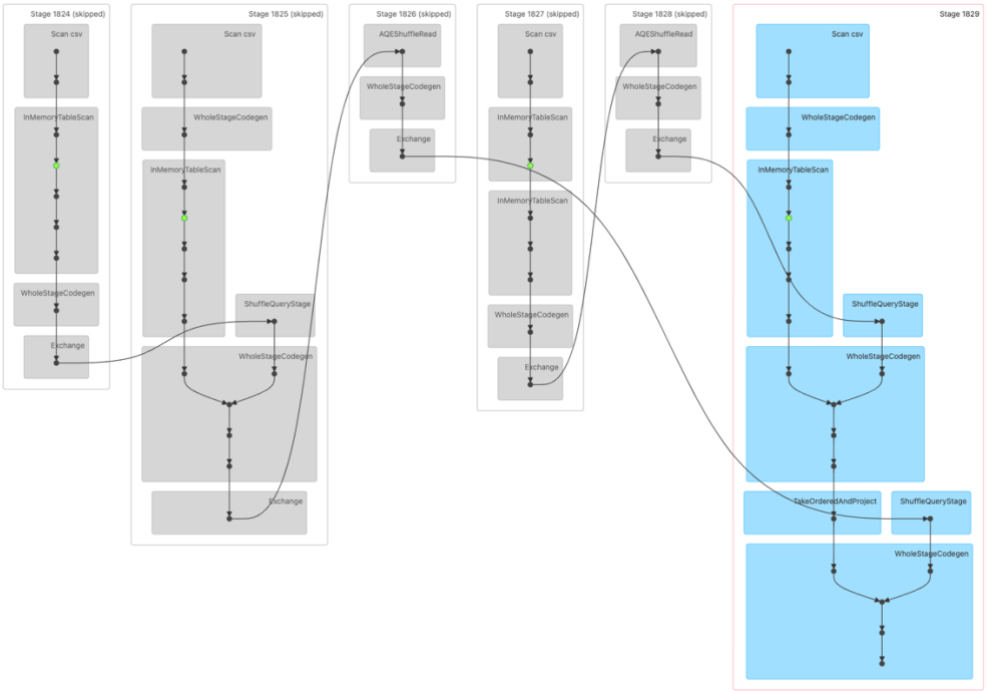
Efficient code ScanCSV runtime 2.9 min

Inefficient code ScanCSV runtime 5.0 min

Furthermore, the placement of WholeStageCodegen is also different in the two codes. In the inefficient code, WholeStageCodegen is independent of ScanCSV, while in the efficient code, WholeStageCodegen is in the same stage as ScanCSV. This means that the efficient code does not need to move between stages when executing ScanCSV and subsequent operations, thus avoiding overhead.

DAG for efficient code

Team contribution	Question 1	Question 2	Report writing
Dawei Yun	Full contributor	Full contributor	Full contributor
Jinpeng Han	No participation	No participation	No participation



DAG for inefficient code