Report

1. Load data

First of all, we have to load the data into all three systems (Pandas, SQLlite, DuckDB). I implement the sql loading part of DuckDB and load data into table directly from the parquet file.

2. Queries

2.1 Query 1

For query 1, I use the SQL function 'COUNT(DISTINCT <column name>)' for each column to count the distinct values for the SQLite and DuckDB. After execute the code we get same resultd of three systems. And then I get the execution time for all the systems.

Tab 1. Execution time for query 1

Name	Times(s)
Pandas	4.059165954589844
SQLite	54.050997495651245
DuckDB	4.369325160980225

Pandas and DuckDB use similar time and SQLLite uses the most time. The main reason for this difference of three systems line in the structure of these three systems.

Pandas is fast because the columns are stored in the NumPy arrays. When we use function "nunique()" of each column, the DataFrame only need to go through the single array instead of the whole table which is similar as a column-oriented database.

SQLite is a traditional row-based database, and it will go through the whole table each time it run the function 'COUNT(DISTINCT <column name>)' which will results in the much more execution time than the other two systems.

DuckDB is mainly benefit from the column-oriented database structure, and it will perform highly efficiently on column search.

2.2 Query 2

For query 2, I use the subquery for SQLite and DuckDB. I also get the same results of all the three systems. The execution time is as follows.

Tab 2. Execution time for query 2

Name	Times(s)
Pandas	1.239544153213501
SQLite	17.60347056388855
DuckDB	0.4161984920501709

For Pandas, it has relatively short execution time. It still benefits from the data structure and in-memory execution. But it takes more time to transform time type to datetime.

For SQLite, it has the longest execution time. It still suffer from it’s row based storage and the disk I/O consumption.

For DuckDB, it has the shortest execution time. It benefits from the column oriented storage and the optimized engine.

3. Machine Learning

I mainly implement the sample generating process for SQLite and regression calculation, sample generating process for DuckDB.

3.1 Implementation

For sample generating process, I mainly use the function ‘RANDOM()’ to sample the random data from the table. I choose random data from the whole table instead of the filtered dataset which is the same way as the Pandas sampling. For test dataset, we shouldn’t use the cleaned data.

For regression calculation of DuckDB, I just follow the similar steps in SQLite. But I use the subquery in the “where” condition to get the standard error first by the function ‘STDDEV_SAMP()’. And then get the beta and average value of fare_amount and trip_distance.

3.2 Execution time

The execution time of three system is as follows.

Tab 3. Execution time for Machine Learning

Name	Times(s)
Pandas	2.867133617401123
SQLite	14.662589073181152
DuckDB	0.773068904876709

For Pandas, it has relatively short execution time. The training process mainly regards the column based operation including multiple, minus, division. Pandas DataFrame can easily manipulate these operations. But the process consists of several steps of such calculations and will cost relatively more time than DuckDB.

For SQLite, it has the longest execution time. It has to run two different SQL because of the lack of the standard deviation function. And the column average and sum calculation also costs more time for row-based database.

For DuckDB, it has the shortest execution time. It benefits from the column oriented storage and the optimized engine. The SQL mainly uses the column calculation which will make it very easy for DuckDB to optimize the execution and get high performance.