FACULTY OF SCIENCE - Course 2019/2020

Advance Data Management for Data Analysis

2nd Assignment NYC Cab dataset ML

Authors:
Lisa Dombrovskij (s1504819)
Margherita Grespan (s2233150)
Juan de Santiago Rojo (s2302470)

1 Introduction

In this assignment, some experiments have been done using Pandas, SQLite and DuckDB, all with the provided data in NYC Cab dataset from 2016. This dataset includes information like pickup and drop-off times, number of passengers, trip distance and fares of cab trips done in New York City in 2016. The three tools are to be compared, in terms of run time and, with the use of machine learning, predictions.

In section 2 the DuckDB implementation is described. In section 3, the two queries for SQLite and DuckDB are developed based on the already given Pandas implementation. Here, the performances of the three are compared. Finally, in section 4, a simple machine learning algorithm is developed in SQLite and DuckDB to predict fare costs. Here, a full implementation in Pandas was given and a partial implementation in SQLite. These were used as a reference for developing the machine learning algorithm with DuckDB. The performance of the three will be shown and described, after the experiments were run.

2 DuckDB Implementation

As most of the code was given, the loading of the data with DuckDB required just a simple line, which is:

```
duck_cursor.execute("COPY yellow_tripdata_2016_01 FROM 'yellow_tripdata_2016-01.csv' HEADER")
```

By doing so, the CSV file containing the dataset is created as a table. Note that because the CSV had a header, the HEADER instruction had to be set. When it comes to loading times, the run time for Pandas loading was 61.34 seconds, whereas with DuckDB, it was 47.51 seconds.

3 SQLite and DuckDB Queries

The next task was to perform two queries on the dataset. The queries are already implemented in Pandas, so that serves as an example. First, the number of distinct (unique) values for each column have to be obtained. This is done with the following query (Q1):

```
""" SELECT
COUNT(DISTINCT VendorID), COUNT(DISTINCT tpep_pickup_datetime), COUNT(DISTINCT
    tpep_dropoff_datetime), COUNT(DISTINCT passenger_count), COUNT(DISTINCT
    trip_distance), COUNT(DISTINCT pickup_longitude), COUNT(DISTINCT pickup_latitude),
    COUNT(DISTINCT RatecodeID), COUNT(DISTINCT store_and_fwd_flag),
    COUNT(DISTINCT dropoff_longitude), COUNT(DISTINCT dropoff_latitude),
    COUNT(DISTINCT payment_type), COUNT(DISTINCT fare_amount), COUNT(DISTINCT
    extra), COUNT(DISTINCT mta_tax), COUNT(DISTINCT tip_amount), COUNT(DISTINCT
    tolls_amount), COUNT(DISTINCT improvement_surcharge), COUNT(DISTINCT total_amount)
FROM yellow_tripdata_2016_01 AS distinct_count """
```

Which results in the following counts respectively: 2, 2368616, 2372528, 10, 4513, 35075, 62184, 7, 2, 53813, 87358, 5, 1878, 35, 16, 3551, 940, 7, 11166. Table 1 shows the counts done by each one of the queries. Note that all three worked successfully.

The second query (Q2) has to find the average, maximum and minimum frequency of events grouped by day and hour. This is done using the following query in SQLite:

```
"""SELECT MIN(cnt), AVG(cnt), MAX(cnt) FROM (SELECT VendorID, tpep_pickup_datetime,
    COUNT(VendorID) cnt FROM yellow_tripdata_2016_01 GROUP BY strftime('\%j',
    tpep_pickup_datetime), strftime('\%H', tpep_pickup_datetime))"""
```

and this subsequent one in DuckDB:

```
"""SELECT MIN(cnt), AVG(cnt), MAX(cnt) FROM (SELECT count(*) as cnt FROM
    yellow_tripdata_2016_01 GROUP BY EXTRACT(DOY FROM tpep_pickup_datetime),
    EXTRACT(HOUR FROM tpep_pickup_datetime)) as cn"""
```

The result of both of these queries matches the result of the Pandas implementation. The minimum is 7, the average is 14659.8 and the maximum is 28511. Table 1 shows each one of the queries' run time.

	Runtime (s)	
	Q1	Q2
Pandas	7.05	13.28
SQLite	78.53	57.55
DuckDB	6.58	0.57

Table 1: Runtime for Query 1 and 2

4 ML Algorithm

As a final task, we implement a machine learning algorithm in DuckDB to predict fare costs based on a given Pandas implementation and a partial one for SQLite. The performance of the three is described here. Note that the implementation and method can be found in the provided code.

With alpha equal to 4.6516 and a beta equal to 2.6614 the runtime of the ML regression for Pandas is 4.226 seconds. Whereas for the SQLite regression this is 141.8774 seconds. Finally, DuckDB's runtime was 1.4604 seconds.

5 Conclusion

In Table 1 it is evident how DuckDB is faster than Pandas and SQLite. On heavy operations, such as `COUNT DISTINCT`, DuckDB is much faster (specifically 12 times faster) than SQLite. However, it is almost equal to the Pandas library. For query 2, even with an aggregation query, DuckDB shows its speed; This time it is 100 times faster.

When it comes to running machine learning algorithms such as linear regression, one can see that the best performing query (in running time) is the one executed with DuckDB, followed by Pandas and then SQLite, being significantly worse than the previously mentioned ones. The accuracy of each one of the proposed methods increases, being Pandas the less accurate when predicting fare rates. Notice that because DuckDB includes a function that can calculate the standard deviation, the regression can be done with just one query. As the figures plotted inside the Jupiter Notebook, the most accurate regression model is the one using DuckDB.

DuckDB is specifically built for the optimization of data science tasks. SQLite is row-based, which makes it suitable when operations have to be performed on a set of rows. However, when it comes to data analysis, one often wants to perform vector-based operations on a subset of columns. Specifically for this reason, and as shown in this report, DuckDB is especially useful.