SIG731 2023: Task 6D

pandas vs SQL

Last updated: 2023-11-24

Contents

Tasks 5–8 are not obligatory; you can submit them in any order (or decide not to tackle them at all). C/D/HD is merely a subjective estimate of their difficulty level. For each task that you successfully complete, you score 10 points (and for those that are not 100% correct, no points will be given).

1 Task

You will study the dataset called nycflights13. It gives information about all 336,776 flights that departed in 2013 from the three New York (in the US) airports (EWR, JFK, and LGA) to destinations in the United States, Puerto Rico, and the American Virgin Islands.

Download the following data files from our unit site (*Learning Resources → Data*):

- nycflights13_flights.csv.gz – flights information,
- nycflights13_airlines.csv.gz – decodes two letter carrier codes,
- nycflights13_airports.csv.gz – airport data,
- nycflights13_planes.csv.gz – plane data,
- nycflights13_weather.csv.gz – hourly meteorological data for LGA, JFK, and EWR.

Refer to the comment lines in the CSV files (note that they are gzipped) for more details about each column.

You aim is to use **pandas** to come up with results equivalent to those that correspond to example SQL queries.

1. Establish a connection with a new SQLite database on your disk.
2. Export all the CSV files to the said database.
3. For each of the SQL queries below (each query in a separate section), write the code that yields equivalent results using **pandas** only and explain – in your own words – what it does.

```
task1_sql = pd.read_sql_query("""
    ...an SQL statement...
""", conn)
```

```
task1_my = (
    ...your solution using pandas... – without SQL
)
pd.testing.assert_frame_equal(task1_sql, task1_my)  # we expect no error here
```

> **Important.** Sometimes, the results generated by **pandas** will be the same up to the reordering of rows. In such a case, before calling `assert_frame_equal`, we should `sort_values` on both data frames to sort them with respect to 1 or 2 chosen columns.

Here are the SQL queries:

1. ```
   SELECT DISTINCT engine FROM planes
   ```

2. ```
   SELECT DISTINCT type, engine FROM planes
   ```

3. ```
   SELECT COUNT(*), engine FROM planes GROUP BY engine
   ```

4. ```
   SELECT COUNT(*), engine, type FROM planes
   GROUP BY engine, type
   ```

5. ```
   SELECT MIN(year), AVG(year), MAX(year), engine, manufacturer
   FROM planes
   GROUP BY engine, manufacturer
   ```

6. ```
   SELECT * FROM planes WHERE speed IS NOT NULL
   ```

7. ```
   SELECT tailnum FROM planes
   WHERE seats BETWEEN 150 AND 210 AND year >= 2011
   ```

8. ```
   SELECT tailnum, manufacturer, seats FROM planes
   WHERE manufacturer IN ("BOEING", "AIRBUS", "EMBRAER") AND seats>390
   ```

9. ```
   SELECT DISTINCT year, seats  FROM planes
   WHERE year >= 2012 ORDER BY year ASC, seats DESC
   ```

10. ```
    SELECT DISTINCT year, seats  FROM planes
    WHERE year >= 2012 ORDER BY seats DESC, year ASC
    ```

11. ```
    SELECT manufacturer, COUNT(*) FROM planes
    WHERE seats > 200 GROUP BY manufacturer
    ```

12. ```
    SELECT manufacturer, COUNT(*) FROM planes
    GROUP BY manufacturer HAVING COUNT(*) > 10
    ```

13. ```
    SELECT manufacturer, COUNT(*) FROM planes
    WHERE seats > 200 GROUP BY manufacturer HAVING COUNT(*) > 10
    ```

14. ```
    SELECT manufacturer, COUNT(*) AS howmany
    FROM planes
    ```

```
GROUP BY manufacturer
ORDER BY howmany DESC LIMIT 10
```

15.
```
SELECT
    flights.*,
    planes.year  AS plane_year,
    planes.speed AS plane_speed,
    planes.seats AS plane_seats
FROM flights LEFT JOIN planes ON flights.tailnum=planes.tailnum
```

16.
```
SELECT planes.*, airlines.* FROM
(SELECT DISTINCT carrier, tailnum FROM flights) AS cartail
INNER JOIN planes ON cartail.tailnum=planes.tailnum
INNER JOIN airlines ON cartail.carrier=airlines.carrier
```

17.
```
SELECT
    flights2.*,
    atemp,
    ahumid
FROM (
    SELECT * FROM flights WHERE origin='EWR'
) AS flights2
LEFT JOIN (
    SELECT
        year, month, day,
        AVG(temp) AS atemp,
        AVG(humid) AS ahumid
    FROM weather
    WHERE origin='EWR'
    GROUP BY year, month, day
) AS weather2
ON flights2.year=weather2.year
AND flights2.month=weather2.month
AND flights2.day=weather2.day
```

Do not include full outputs of the SQL queries in the report.

Do not forget to call `pd.testing.assert_frame_equal` in each case.

Do not forget to explain each query in your own words.

2 Artefacts

At the start of the notebook, you need to provide: the **title** of the report (e.g., *Task 42: How Much I Love This Unit*), your **name**, **student number**, and **email address**.

Then, add 1–2 introductory paragraphs (an introduction/abstract – what the task is about).

Before each nontrivial code chunk, briefly **explain** what its purpose is. After each code chunk, **summarise and discuss the obtained results** (in a few sentences).

Conclude the report with 1–2 paragraphs (summary/discussion/possible extensions of the analysis etc.).

Checklist:

1. Header, introduction, conclusion (Markdown chunks).
2. Text divided into sections, all major code chunks commented and discussed in your own words (Markdown chunks).
3. Every subtask addressed/solved. In particular, all reference results that are part of the task specification have been reproduced (plots, computed aggregates, etc.).
4. The report is readable and neat. In particular:
    - all code lines are visible in their entirety (they are not too long),
    - code chunks use consecutive numbering (select *Kernel - Restart and Run All* from the Jupyter menu),
    - rich Markdown formatting is used (`# Section Title`, `* bullet list`, `1. enumerated list`, `| table |`, `*italic*`, etc.),
    - the printing of unnecessary/intermediate objects is minimised (focus on reporting the results specifically requested in the task specification).

Submissions which do not *fully* (100%) conform to the task specification *on* the cut-off date will be marked as FAIL.

Good luck!