SIT220/731 2024.T3: Task 5C

pandas vs SQL

Last updated: 23rd November 2024

Contents

> Tasks 5–8 are not obligatory, you can decide not to tackle them at all. C/D/HD is merely a subjective estimate of their difficulty level.

This task is related to Module 1.6 - 1.10; see the *Learning Resources* on the unit site.

This task is due on **Week 11 (Sunday)**. Start tackling it as early as possible. If we find your first solution incomplete or otherwise incorrect, you will still be able to amend it based on the generous feedback we will give you. In case of any problems/questions, do hot hesitate to attend our on-campus/online classes or use the Discussion Board on the unit site.

Submitting after the aforementioned due date will incur a late penalty.

All submissions will be checked for plagiarism. You are expected to work independently on your task solutions. Never share/show parts of solutions with/to anyone.

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.

Just like in Module 1.10 (Database Access), your aim is to use **pandas** to come up with results equivalent to those that correspond to example SQL queries.

Important. This time, you will generate the Jupyter/IPython notebook using jupytext. Write all your code in a single .py file (Python source script; a plain text file) in the `py:percent` format, and define Markdown chunks therein using a special syntax (see the jupytext documentation for more details). Pair this source file with a notebook in the .ipynb format (how to do that you will have to learn yourself – this is part of this C-level task). You will submit both files via OnTrack.

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
      ```

```sql
WHERE year >= 2012 ORDER BY year ASC, seats DESC
```

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

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

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

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

14.
```sql
SELECT manufacturer, COUNT(*) AS howmany
FROM planes
GROUP BY manufacturer
ORDER BY howmany DESC LIMIT 10
```

15.
```sql
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.
```sql
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
```

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 Additional Tasks for Postgraduate (SIT731) Students (*)

Postgraduate students, apart from the above tasks, are additionally **required** to solve/address/discuss what follows.

17. An additional SQL query to implement:

```sql
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
```

3 Optional Features (**)

As only practice makes perfect, you may be interested in solving the following *additional* tasks (no extra points, though):

1. Use the `timeit` or `time` module to compare the run-times of SQLite3 (through **pandas**) vs pure **pandas** solutions.
2. If there are many ways to implement a given operation, do not hesitate to list possible alternatives.
3. Implement all the queries using the **polars** package, which is being developed as an alternative to **pandas** (see, e.g., the discussion here). Note that the package is still a work in progress. Compare the run-times (see item 1 above).

4 Artefacts

Make sure that your notebook has a **readable structure**; in particular, that it is divided into sections. Use rich Markdown formatting (text in dedicated Markdown chunks – not just Python comments).

Do not include the questions/tasks from the task specification. Your notebook should read nicely and smoothly – like a report from data analysis that you designed yourself. Make the flow read natural (e.g., *First, let us load the data on... Then, let us determine... etc.*). Imagine it is a piece of work that you would like to show to your manager or clients — you certainly want to make a good impression. Check your spelling and grammar. Also, use formal language.

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**, **email address**, and whether you are an **undergraduate (SIT220) or postgraduate (SIT731)** student.

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.).

Limitations of the OnTrack ipynb-to-pdf renderer:

Ensure that your report as seen in OnTrack is aesthetic (see *Download submision PDF* after uploading the .ipynb file). The OnTrack ipynb-to-pdf renderer is imperfect. We work with what we have. Here are the most common Markdown-related errors.

- Do not include any externally loaded images (via the `![label](href)` Markdown command), for they lead to upload errors.

- Do not input HTML code in Markdown.

- Make sure you leave one blank line before and after each paragraph and bullet list. Do not use backslashes at the end of the line.

- Currently, also *LaTeXformulae* and Markdown tables are not recognised. However, they do not lead to any errors.

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!

5 Intended Learning Outcomes

ULO	Is Related?
ULO1 (Data Processing/Wrangling)	YES
ULO2 (Data Discovery/Extraction)	YES
ULO3 (Requirement Analysis/Data Sources)	YES
ULO4 (Exploratory Data Analysis)	
ULO5 (Data Privacy and Ethics)	