CSED226 Intro. to Data Analysis : Assn2 Report

September 27, 2025

Student 20240505 Hyunseong Kong

1. Overview

The second assignment is to be familiar with the advanced pandas technique by processing the real-world NYC Flights 2013 dataset. The task is sequential, requiring data loading, cleaning, outlier detection, grouping/aggregation, reshaping, window functions, and complex filtering.

2. Differences

Before starting, the submitted code attempted to process all operations in a single pipeline as much as possible. In contrast, the solution code defined multiple variables, separating the code into parts.

2-1. Task 1: Data Loading, Type Conversion, and Initial Cleaning

Instructions:

1. Convert `dep_time` and `arr_time` to hours-of-day using vectorized ops.
2. Make `carrier` and `origin` categorical.
3. Drop every row that contains NaN in `arr_delay`.
4. Reset index of the DataFrame.

According to the assignment specifications, the time was provided in HHMM format. I interpreted this as needing to split it into HH (hours) and MM (minutes) for calculation. However, the solution code interpreted the entire time value as minutes. Since the HHMM format implies that 0930 should be interpreted as 9:30, I believe this is an error in the solution code. Other than that, there are no significant differences between the solution and submitted code. A minor difference is the label name of the converted time column, which is not considered a major issue.

2-2. Task 2: Outlier Detection and Robust Cleaning

Instructions:

1. Use IQR to detect outliers in `arr_delay` (lower = Q1 - 1.5×IQR, upper = Q3 + 1.5×IQR).
2. Add `delay_category` using pd.cut. (bins=[$-\infty$, 15, 60, ∞], labels=['short', 'medium', 'long'])

Both the solution and submitted code used bitmasks for filtering. The `quantile` function was used identically to calculate the IQR, and `df_filtered` was copied to calculate `outliers_removed`. The `delay_category` was also calculated in the same way. Therefore, there are no functional differences between the two codes.

2-3. Task 3: Advanced Grouping and Custom Aggregation

Instructions:

1. Group by `origin` and `carrier`.
2. Aggregate: `arr_delay`: mean / `air_time`: median / `distance`: 95th percentile
3. `Proportion long delays`: `lambda (x > 60).mean()`
4. Sort by `mean_arr_delay` desc.

5. Show top 5 rows.

"Table with columns like mean_arr_delay, prop_long_delay, median_air_time, p95_distance."

Because the 'Expected' section of the assignment contained the above text, the submitted code used these names when performing the aggregation. Additionally, referencing page 25 of the Pandas II PPT, the submitted code used named-agg syntax for all aggregations to avoid a multi-index. The solution code performed rounding, but this was not a requirement in the assignment, so it was not done in the submitted code. The final requirement to show 5 rows was already handled by the print function, so no extra code was added.

2-4. Task 4: Reshaping and Pivot Analysis

Instructions:

1. Pivoting the table by setting rows=`month`, cols=`carrier`, values=`mean_dep_delay`. (fill 0, margins=True)
2. Melt the pivotted table back to long format. Add metric=`mean_dep_delay` column.
3. Filter to top 3 `carriers` by flight count.
4. Compute correlation between `mean_dep_delay` and `month`.

In both codes, the pivoted table was calculated with the same parameters. However, in the submitted code, `margins=True` was added as it is a requirement of the assignment, which necessitated adding code later to delete the 'All' row and column. Both codes calculated the top 3 carriers in a similar manner. However, the solution code filtered for the top 3 carriers first, creating `pivot_top`, and then used only this subset for the melt operation. The submitted code, on the other hand, calculated all values first and then filtered for the top 3 carriers. In this respect, the solution code's approach is considered better.
The line `.assign(metric='mean_dep_delay')` in the submitted code was included by mistake. This was a human error resulting from a misinterpretation of the instruction "Add metric=`mean_dep_delay` column". Since the 'metric' column was ultimately not used, it didn't cause a major issue with the correlation calculation, but an unnecessary column was unintentionally added.
The correlation calculation was performed similarly, but the submitted code selected two columns and then executed the `.corr()` function. Since the result of this calculation is a correlation matrix, the `iloc` function was used to extract the desired value.

2-5. Task 5: Window Functions and Ranking

Instructions:

1. Rank carriers by `mean_arr_delay` (asc) within each month-origin partition.
2. Compute cumulative avg `arr_delay` per `carrier` over months (expanding mean).
3. Find worst `carrier` (highest mean delay) per `month`.

The behavior of the solution code and the submitted code differs for this problem. The submitted code interpreted 'worst' literally, finding the carrier with the worst performance for each month. In contrast, the solution code calculated an average value. Since the intended method for the task was to use the average, the submitted code cannot be considered to have solved the problem correctly.
On the other hand, for the 'top 3', the submitted code correctly calculated results that met the objective of finding the top 3 carriers with the longest cumulative average delay time by month. The submitted code calculates the average arrival delay `arr_delay`, sorts it by carrier and month for cumulative calculation, groups by carrier, and then takes the last value of the `cum_avg` column (the final cumulative average). Finally, it creates a `rank` column to determine the final ranking and then filters for the top 3 carriers.

2-6. Task 6: Complex Filtering and Derived Insights

Instructions:

1. Filter: `distance` $>$ 1000 & `air_time` $>$ 90th percentile.
2. Compute `score`: (`dep_delay` + `arr_delay`)/`distance` $\times$ 100.

3. Group by `dest`: mean score, count.
4. Filter `count` $>$ 1000.
5. Sort mean score in descent.
6. Top 5 worst dests.

Overall, the code was written almost identically. The filtering and `score` calculation were performed successfully. However, in the aggregation, the count was not explicitly specified, so it was calculated based on the `score` column. This affected the subsequent calculation of `d['count'] > 1000`. This was a human error that occurred from copying the earlier part of the aggregation.

3. Lessons Learned

Through this assignment, I realized that while writing a pipeline in a single line might look good, it reduces code readability and can lead to unintentional mistakes. Therefore, when using pandas, I should separate each part and complete its functionality individually. Also, I had some problems in the code because I didn't interpret the instructions properly. I would have implemented each one correctly if I hadn't made a mistake, but I need to be careful because such a slight change in the code makes a big difference.