DataEng S24: Data Validation Activity

Deepak Lukulapu

High quality data is crucial for any data project. This week you'll gain experience with validating a real data set provided by the Oregon Department of Transportation.

Due: this Friday at 10pm PT
Submit: Make a copy of this document and use it to record your results. Store a PDF copy of the document in your git repository along with any needed code before submitting using the in-class activity submission form.

A. [MUST] Initial Discussion Question

Discuss the following question among your working group members at the beginning of the week and place your own response(s) in this space. Or, if you have no such experience with invalid data then indicate this in the space below.

Have you ever worked with a set of data that included errors? Describe the situation, including how you discovered the errors and what you did about them.

Response:

No, I do not have such experience with invalid data.

Background

The data set for this week is a listing of all Oregon automobile crashes on the Mt. Hood Hwy (Highway 26) during 2019. This data is provided by the Oregon Department of Transportation and is part of a larger data set that is often utilized for studies of roads, traffic and safety.

Here is the available documentation for this data: description of columns, Oregon Crash Data Coding Manual

Data validation is usually an iterative multi-step process.

B. Create assertions about the data
C. Write code to evaluate your assertions.

D. Run the code, analyze the results
E. Write code to transform the data and resolve any validation errors

B. [MUST] Create Assertions

Access the crash data, review the associated documentation of the data (ignore the data itself for now). Based on the documentation, create English language assertions for various properties of the data. No need to be exhaustive. Develop one or two assertions in each of the following categories during your first iteration through the ABC process.

1. *existence* assertions. Example: “Every crash occurred on a date”
2. *limit* assertions. Example: “Every crash occurred during year 2019”
3. *intra-record* assertions. Example: “If a crash record has a latitude coordinate then it should also have a longitude coordinate”
4. Create 2+ *inter-record check* assertions. Example: “Every vehicle listed in the crash data was part of a known crash”
5. Create 2+ *summary* assertions. Example: “There were thousands of crashes but not millions”
6. Create 2+ *statistical distribution assertions*. Example: “crashes are evenly/uniformly distributed throughout the months of the year.”

These are just examples. You may use these examples, but you should also create new ones of your own.

ANS:

Existence Assertions

1. **Every crash record must include a non-null 'Crash ID'** - Ensures that each crash is uniquely identifiable.
2. **Every crash record must have a valid 'Crash Date'.** - Confirms that no crash entry is missing a date.

Limit Assertions

1. **Every crash occurred during the year 2019.** - Ensures all crash records are from the specified year, crucial for annual reporting and analysis.
2. **The 'Crash Hour' field should only contain values from 0 to 23.** - Validates that the hours are recorded on a 24-hour clock.

Intra-record Assertions

1. **If a crash record has a 'Latitude' coordinate, it must also have a 'Longitude' coordinate.** - Ensures geographic data completeness.
2. **If 'Alcohol Involved' is marked 'Yes', there should be an alcohol level recorded.** - Checks for consistency in reporting substance involvement.

Inter-record Check Assertions

1. **Every vehicle ID listed in crash records must correspond to an existing 'Crash ID'.** - Ensures all vehicles are linked to a reported crash.
2. **Participant IDs in vehicle records should match those in participant details records.** - Ensures consistency in participant data across different records.

Summary Assertions

1. **There were thousands of crashes but not millions in the dataset.** - A broad check on the scale of the data, useful for catching data loading errors.
2. **The number of crashes involving alcohol should not exceed the total number of crashes.** - Validates that the subset counts are within logical limits.

Statistical Distribution Assertions

1. **Crashes are evenly distributed across all months of the year.** - Checks for uniformity in the distribution, which could inform about seasonal impacts.
2. **The average number of participants per crash should be less than 10.** - Ensures that the data remains realistic; very high averages could indicate data errors or duplicates.

C. [MUST] Validate the Assertions

1. Study the data in an editor or browser. Study it carefully, this data set is non-intuitive!.
2. Write python code to read in the test data. You are free to write your code any way you like, but we suggest that you use pandas' methods for reading csv files into a pandas Dataframe.
3. Write python code to validate each of the assertions that you created in part A. The pandas package eases the task of creating data validation code.

4. If needed, update your assertions or create new assertions based on your analysis of the data.

D. [MUST] Run Your Code and Analyze the Results

Results:

Passed: Every record has a Crash ID
Passed: Every Latitude has corresponding Longitude
Passed: There were thousands of crashes but not millions
Passed: The number of crashes involving alcohol does not exceed total crashes

In this space, list any assertion violations that you encountered:

- Failed: Every record has a Crash Day
- Failed: All crashes occurred during the year 2019
- Failed: Crash Hour within valid range (0-23)
- Failed: Crashes are evenly distributed throughout the months of the year
- Failed: Every vehicle listed was part of a known crash

For each assertion violation, describe how to resolve the violation. Options might include:

- ❖ revise assumptions/assertions
- ❖ discard the violating row(s)
- ❖ Ignore
- ❖ add missing values
- ❖ Interpolate
- ❖ use defaults
- ❖ abandon the project because the data has too many problems and is unusable

Assertion Failure on 'Crash Day' Validity

- ➢ Issue: Not all records contain a valid 'Crash Day'. This may be due to missing values or incorrect date formats.
- ➢ Proposed Solutions:
  - ■ Discard Violating Rows: Remove entries without critical date information, assuming the dataset size allows for some data loss without impacting the overall analysis.
  - ■ Interpolate Dates: Estimate missing dates basd on surrounding data points where feasible.

Assertion Failure on Crash Year

- ➢ Issue: Some records feature dates not within the year 2019.
- ➢ Proposed Solutions:
  - ■ Revise Assertions: If data from years other than 2019 are acceptable for our study, I suggest we adjust the assertion accordingly.
  - ■ Remove Outliers: Strictly focus on 2019 data by removing records from other years.

Assertion Failure on 'Crash Hour'

- ➢ Issue: Some entries have 'Crash Hour' values outside the 0-23 range, indicating potential data entry errors or misformats.
- ➢ Proposed Solutions:
  - ■ Set Defaults: Assign a default value such as the median hour to out-of-range entries, though this may skew our analysis.
  - ■ Discard Violating Rows: Remove these entries if they are few and correcting them isn't feasible.

Assertion Failure on Vehicle Crash Linkage

- ➢ Issue: There are vehicle IDs that do not match any recorded crash IDs, suggesting possible issues in data linkage or entry errors.
- ➢ Proposed Solutions:
  - ■ Revise Data Linkage: Check and correct the linkage between vehicle IDs and crash IDs.
  - ■ Remove Inaccurate Entries: Eliminate entries with unverifiable vehicle IDs.

Assertion Failure on Monthly Crash Distribution

- ➢ Issue: Crash occurrences vary significantly by month, which could reflect true seasonal trends or inconsistencies in data collection.
- ➢ Proposed Solutions:
  - ■ Revise Assumptions: Recognize and incorporate seasonal variation into our analysis model.
  - ■ Ignore Variation: If this uneven distribution aligns with expected seasonal effects and does not detract from our study's objectives, we might opt to disregard this variation.

No need to write code to resolve the violations at this point, you will do that in step E.

E. [SHOULD] Resolve the Violations and Transform the Data

For each assertion violation write python code to resolve the violation according to your entry in the “how to resolve” section above.

Output the validated/transformed data to new files. There is no need to keep the same, awkward, single file format for the data. Consider outputting three files containing information about (respectively) crashes, vehicles and participants.

F. [ASPIRE] Learn and Iterate

The process of validating data usually gives us a better understanding of any data set. What have you learned about the data set that you did not know at the beginning of the current ABC iteration?

Next, iterate through the process again by going back through steps B, C, D and E at least one more time.