DataEng: Data Validation Activity

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 for this week.

High quality data is crucial for any data project. This week you'll gain some experience and knowledge of analyzing data sets for quality.

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 three-step process. First (part A) you develop assertions about your data as a way to make your assumptions explicit. Second (part B) you write code to evaluate the assertions and test the assumptions. This helps you to refine your existing assertions (part C) before starting the whole process over again by creating new assertions (part A again).

Submit: In-class Activity Submission Form

A. 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 for this assignment, two or more assertions in each category are enough.

1. Create 2+ existence assertions. Example, "Every record has a date field".
    a. "Every record has a crash serial number"
    b. "Every record has a crash hour"

2. Create 2+ limit assertions. The values of most numeric fields should fall within a valid range. Example: "the date field should be between 1/1/2019 and 12/31/2019 inclusive"
    a. "The crash hour field should be between 00-23 inclusive or is 99"
    b. "The county code is between 01 and 36 inclusive"
    c. "The city value should be between 001 and 250 inclusive, or is blank"

3. Create 2+ intra-record check assertions.
    a. “If the month is February, the day of the month must be less than 30”
    b. Total Pedestrian Fatality Count can’t exceed Total Fatality Count

4. Create 2+ inter-record check assertions. [these might be summaries, but I can’t come up with anything else]
    a. The participant id of a crash should not be the same as next entry
    b. “The total TOT_DRVR_AGE_01_20_CNT can’t exceed the total TOT_PER_INVLV_CNT”

5. Create 2+ summary assertions. Example: “every crash has a unique ID”
    a. All crashes probably shouldn’t have more than 100 fatalities each
    b. Each crash has a unique DMV crash serial number

6. Create 2+ referential integrity insertions. Example “every crash participant has a Crash ID of a known crash”
    a. Every crash id in the participants table should refer to a crash id in the crashes table (once the data is split up into the corresponding tables)
    b. Every crash id in the vehicles table should refer to a crash id in the crashes table

7. Create 2+ statistical distribution assertions. Example: “crashes are evenly/uniformly distributed throughout the year.”
    a. The crashes should not be linearly distributed across all times of the day
    b. The crashes should not be linearly distributed across all times of the year (during holidays, we might expect more crashes than usual)

B. Validate the Assertions

1. Now study the data in an editor or browser. If you are anything like me you will be surprised with what you find. The Oregon DOT made a mess with their data!

2. Write python code to read in the test data and parse it into python data structures. You can 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. Again, pandas makes it easy to create and execute assertion validation code.

    For the sake of time, I’ve skipped the inter-record check assertions

4. If you are like me you'll find that some of your assertions don't make sense once you actually understand the structure of the data. So go back and change your assertions if needed to make them sensible.

5. **Run your code and note any assertion violations. List the violations here.**

   The violations were that:
   1. There were entries with duplicate crash serial numbers
   2. There were entries with city section IDs that weren't in the propper range (1 to 250).

C. Evaluate the Violations

For any assertion violations found in part B, describe how you might resolve the violation. Options might include "revise assumptions/assertions", "discard the violating row(s)", "ignore", "add missing values", "interpolate", "use defaults", etc.

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

If you chose to "revise assumptions/assertions" for any of the violations, then briefly explain how you would revise your assertions based on what you learned.

For the "Duplicate Crash Serial Number Entry" violations, I think I would remove all the offending entries. In fact, I might remove all the entries that have the invalid crash serial number. I could try to keep one entry with the offending serial number, but, presuming the serial numbers are supposed to uniquely identify a single crash, it might be more correct to remove them all. I don't know which one contains the right data, after all. Alternatively, I could try to modify the serial number (by appending a 1 to it or some other unique, "we have a problem here" indicator) if the accuracy of the number does not matter that much. If I'm mistaken and the crash serial number is not unique (at least, not in all cases), then the solution would be to revise the duplicate serial number check rule to account for allowed duplicates, or remove it altogether (since it would be wrong).

For the City Section ID Not In Range violations, the solution is to revise my assertion to check the range 0-250 instead of 1-250, because there are cases where the value can be and likely is null (which, if my presumption is correct, translates to a "0.0" value when read in via pandas).

D. Learn and Iterate <- *Stopped Here*

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 ABCD iteration?

Next, iterate through the process again by going back to Step A. Add more assertions in each of the categories before moving to steps B and C again. Go through the full loop twice before moving to step E.

E. Resolve the Violations

For each assertion violation found during the two loops of the process, write python code to resolve the assertions. This might include dropping rows, dropping columns, adding default values, modifying values or other operations depending on the nature of the violation.

Note that I realize that this data set is somewhat awkward and that it might be best to "resolve the violations" by restructuring the data into proper tables. However, for this week, I ask that you keep the data in its current overall structure. Later (next week) we will have a chance to separate vehicle data and participant data properly.

E. Retest

After modifying the dataset/stream to resolve the assertion violations you should have produced a new set of data. Run this data through your validation code (Step B) to make sure that it validates cleanly.

Submit: In-class Activity Submission Form