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".

   VALIDATED - Every record has a Crash ID.

   See below in the intra-record check assertions. ~~*Not sure how to do this in the current format, as each Crash ID has many lines, but only one contains the serial number.~~ - Every record has a DMV Crash Serial Number.

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"

VALIDATED (Note: don't think the 8 digit spec is being checked, only the numerical range) - Every Crash ID is between 00000000 and 99999999 (note: CRASH_ID is a unique 8 digit identifying number). *Not ideal on the low end as we're dealing with the num type (not char) and 00000000 = 0 so doesn't guarantee 8 digits.

VALIDATED (Note: used 1 as low end of range, as 00001 was not allowed as a parameter) - Every DMV Crash Serial Number is a string of characters between 00001 and 99999 inclusive.

VALIDATED - Every Crash Day is a string of characters between 00 and 24 or it is 99.

3. Create 2+ *intra-record check* assertions.

***Crash Date does not exist in CSV file - ~~The Crash Year Number, Crash Month Number, and Crash Day Number should be equivalent to the respective parts of the Crash Date.~~**

***All data is numerical and requires table lookups for many fields, so there is nothing to cross reference in cases like this - ~~The County Code should map to the County Name.~~**

OK, starting to understand the spreadsheet better (and its relationship to the DMV documentation). Here's another try at intra-record checks:

VALIDATED - In cases where Record Type = 1, Crash ID will be unique.
- In cases where Record Type = 1, there will be a Serial # in the range

4. Create 2+ *inter-record check* assertions.

VALIDATED ~~*Don't see how to check this in current format~~ - No two records should have the same Crash ID.

***Serial # / DMV Serial Number is more complicated than I realized. Each crash in the spreadsheet has a Serial #; the DMV Serial Number prepends the County Code, but the DMV Serial Number is not in the spreadsheet. Then there are systems for indicating records where the county was originally misidentified and where duplicate crash serial numbers were assigned. I'm going to bail on this validation - No two records should have**

the same DMV Serial Number, aside from those where it's expected (above 79999 duplicates will exist).

5. Create 2+ *summary* assertions. Example: “every crash has a unique ID”

   VALIDATED - Every record has a total vehicle count.

   - Every record indicates whether alcohol was involved.

6. Create 2+ referential integrity insertions. Example “every crash participant has a Crash ID of a known crash”

   - Every city listed should be a city in Oregon.

   - Every county listed should be a county in Oregon.

7. Create 2+ *statistical distribution assertions*. Example: “crashes are evenly/uniformly distributed throughout the year.”

   - There will be a spike in crashes around holidays such as July 4, Thanksgiving, Christmas, New Year's Eve/Day.

   - There will be a spike in crashes around 2:30 AM, the time bars must close in the state.

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!
   Yes, many rows for each Crash ID!
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.
   OK.
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.
   OK ...
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.
   Yes, this was my experience too.

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

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.
A1

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

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.

D. 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 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