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.

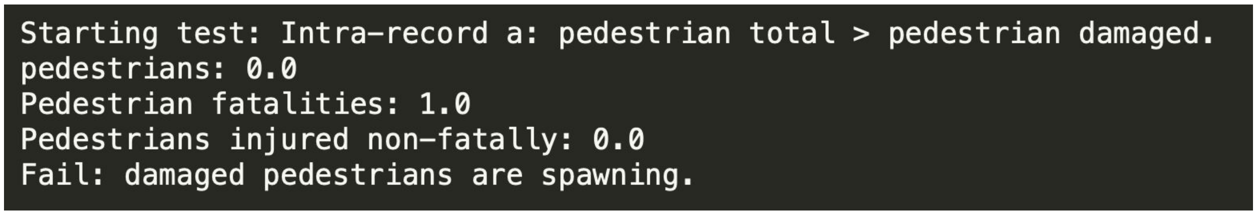
1. Create 2+ *existence* assertions. Example, "Every record has a date field".
    a. Every crash has a vehicle, bicycle, or motorcycle associated.
    b. Each crash has a Serial Number.
2. Create 2+ *limit* assertions. The values of most numeric fields should fall within a valid range. Example: "the date field should be between 1/1/2019 and 12/31/2019 inclusive"
    a. There are 3 record types, each record must belong to one of the types. Record type is in set(1,2,3)
    b. The speed limit is positive and less than 101. 101 > POST_SPEED_LMT_VAL > 0
3. Create 2+ *intra-record check* assertions. Between records of different attributes.

    a. Total pedestrians involved is the sum of all types of pedestrian injuries or non injuries.
       TOT_PED_CNT = sum(TOT_PED_FATAL_CNT, TOT_PED_INJ_LVL_A_CNT, TOT_PED_INJ_CNT)
    b. If the Speed Involved Flag is not null, there is a greater likelihood of fatal injuries TOT_FATAL_CNT. Aka sum(CRASH_SPEED_INVLV_FLG)/ sum(TOT_FATAL_CNT) > 0
4. Create 2+ *inter-record check* assertions. Between records of the same attribute.
    a. The average count of vehicles involved is between 0 and 3.
       3 >= avg(TOT_VHCL_CNT) >= 0
    b. There are crashes every month of the year.
5. Create 2+ *summary* assertions. Example: “every crash has a unique ID”
    a. Every record has a crash ID.
    b. Crash IDs are unique
6. Create 2+ referential integrity insertions. Example “every crash participant has a Crash ID of a known crash”
    a. Each participant has a crash ID and vehicle ID (even if it is 0)
    b. Each vehicle has a crash ID
7. Create 2+ *statistical distribution assertions*. Example: “crashes are evenly/uniformly distributed throughout the year.”
    a. Fatalities as a percentage of crashes is even throughout the year.
    b. Passengers in vehicle has a normal form distribution

B. Validate the Assertions

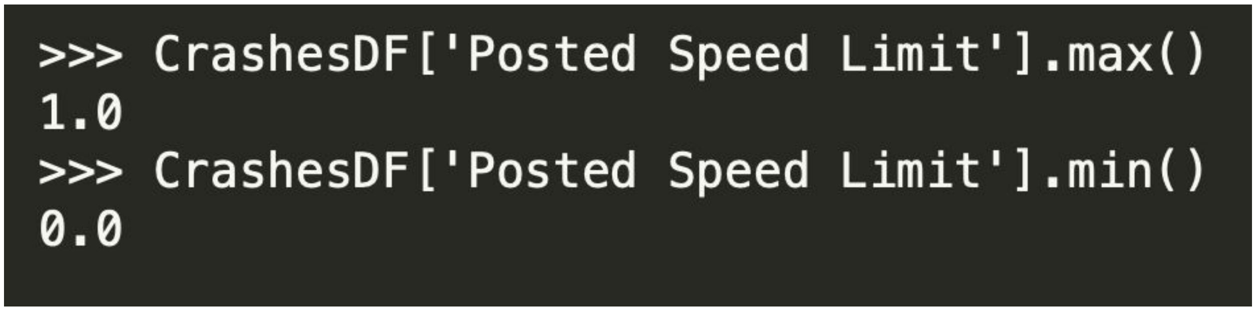
1. Now study the data in an editor or browser. If you are anything like me you will be surprised with what you find. The Oregon DOT made a mess with their data!
2. Write python code to read in the test data and parse it into python data structures. You can write your code any way you like, but we suggest 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.
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.

Intra-Record test a: there were more pedestrian fatalities than there were pedestrians, in at least 1 crash.



```
Starting test: Intra-record a: pedestrian total > pedestrian damaged.
pedestrians: 0.0
Pedestrian fatalities: 1.0
Pedestrians injured non-fatally: 0.0
Fail: damaged pedestrians are spawning.
```

Also even for some checks that did pass, I am learning the values do not represent what I thought they did. These values are within the range of 0-101. However this assertion is invalid given this data.



```
>>> CrashesDF['Posted Speed Limit'].max()
1.0
>>> CrashesDF['Posted Speed Limit'].min()
0.0
```

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 both of the violations above, it indicates I misunderstood what the data represents. I would likely simply replace these assertions with new ones. Potentially for the pedestrian count check, if the row passes all the other checks, I could set the total of pedestrials equal to the sum of those injured and fatally injured. This would fix incorrect data, if most of the failures are as detailed above.

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?

My assumptions about field correlation were incorrect. In places where there are totals, it is often not possible to sum to that total. For example I had assumed I could check total pedestrians as the sum of not injured pedestrians, injured pedestrians, and fatally injured pedestrians. Upon closer inspection, non-injured pedestrians was not recorded. This made me change the test to a greater than / less than relationship.

I assumed the posted speed limit would be the speed limit that is posted. Instead it is a boolean. I'm still investigating what this means.

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.

R. 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