DataEng S23: Data Validation Activity

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.

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.

Initial Discussion Question

- Discuss the following question among your working group members at the beginning of the week and place your responses 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 1: My Response: I have worked with data that had errors during the project for Intro to DBMS course. We used supermarket sales data for the project which had three csv files with redundant data in multiple csv files. Because of the redundant data certain update queries resulted in making some of the records inconsistent. Also, the date fields in the files had format issue. In order to make the data consistent and write meaningful queries we cleaned the data using the excel macros and transformed the csv files into valid consistent data.

Response 2: Shreshtha Siddhi : Yes, I have worked with a set of data that included errors. It was during my work where I was working on Telecommunication data. I discovered the errors because the SQL queries I developed to fetch the data were giving empty output. So, I traced the data flow back up until the source database tables and discovered that some of the values of the primary key column were null. I finally reached out to the source team to verify the data.

Response 3: Shatabdi Pal: I worked with health data for Database project. The original data was in .csv format and some fields were blank. Some of the fields have inconsistent data formatting such mismatches in naming and punctuation marks. That data needed to be cleaned and transformed before loading data into Postgres database schema. Database was throwing error for unique id violation. The querying the database was not giving exact count and not fetching record as there were issues with record malformation.

Response 4: Kavitha Jajula: Yes, I have worked with Datasets in my work experience. The dataset received from client has different column names when compared to the SailPoint tool data which we use for project. It was throwing error when we are trying to because it is not matching with the database field names. Hence we cleaned the data so that all column names matches as per the database and proceeded for certification by uploading data in to SailPoint.

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.

A. Create assertions about the data
B. Write code to evaluate your assertions.
C. Run the code, analyze the results and resolve any validation errors

Repeat this ABC loop as many times as needed to fully validate your data.

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. 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"
   Assertion: "Every crash has highway number as 26"
2. *limit* assertions. Example: "Every crash occurred during year 2019"
   Assertion: "The weekday code for each crash is between 1 to 7"
3. *intra-record* assertions. Example: "If a crash record has a latitude coordinate then it should also have a longitude coordinate"
   Assertion: "If a crash record has a special jurisdiction then it should also have jurisdiction group"
4. Create 2+ *inter-record check* assertions. Example: "Every vehicle listed in the crash data was part of a known crash"
   Assertion: "Every participant mentioned in the data is part of a known crash"
5. Create 2+ *summary* assertions. Example: "There were thousands of crashes but not millions"
   Assertion: "All the crash ids are unique"
6. Create 2+ *statistical distribution assertions*. Example: "crashes are evenly/uniformly distributed throughout the months of the year."
   Assertion: "Crashes are not evenly distributed throughout the day"

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

B. 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.

C. Run Your Code and Analyze the Results

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

- The crash hour column when used to plot the distribution of the crashes during the hours between 0-23 shows that there are some rows with value 99. The value 99 is used when the time of crash is unknown. So, the assertion is validated, however, there are certain unusual rows with value 99.

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

- revise assumptions/assertions
- discard the violating row(s)
- **Ignore:** The rows with crash hour value 99 can be ignored and as the distribution clearly shows that the crashes are not evenly distributed and thus the statistical assertion is validated.
- add missing values
- Interpolate
- use defaults
- abandon the project because the data has too many problems and is unusable

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

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 ABC iteration?

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

E. 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.