DataEng S22: Data Validation Activity

High quality data is crucial for any data project. This week you'll gain experience with validating a real data set.

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: Yes, while working on the huge datasets I found that there are many empty cells and noise present in the data which is not required for visualization purposes. I have discovered the errors as the data was not getting pushed to Kafka servers as it has the different time format. Just ignored and removed the unnecessary noise data which is not required for visualization purposes → Tarun Kumar

Response 2: While working with the large datasets I have observed that there is full of empty spaces and incorrect data and type formats which are not supported while data integration and then we found this would not work and designed a particular format of things which needs to be trimmed and pushed into the data lakes and pushed into the servers in order to be very clear with the headings .
The best resolution we have done is using the particular standard format as per the kafka as per → Pragina Vaidya

Response 3: Don't have any experience with invalid data or data in general → Ilias Bonie

Response 4: Only 3 members. N/A

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”
2. *limit* assertions. Example: “Every crash occurred during year 2019”
3. *intra-record* assertions. Example: “Every crash has a unique ID”
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.

1. Existence assertions:
    a. Every record has a crash ID
2. Limit assertions:
    a. Evey record has a record type from 1 to 3
3. Intra-assertions:
    a. Every crash has a unique serial #
4. Inter-record check assertions (2+):
    a. Each crash ID is associated with at least one vehicle
    b. Each crash ID is assoicated with at least one participart
5. Summary assertions (2+):
    a. Every crash in the record has a NHS flag of 1
    b. On average, crashes involve 2 or more vehicles
    c. Each date had a few/couple crashes reported but not hundreds

6. Statistical distributions assertions (2+):
    a. Crashes occurred have approx the same longitude degrees (-122 to -121)
    b. Crashes are evenly/uniformly distributed throughout the “direction from intersection”

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:

- From the quick glance of the data set, i expected that every dataset had a NHS flag of 1 but that wasn’t the case here. Some had 0. In order to fix this, I would revise my assumption about the data
- From my observations I expected every dataset to have a unique serial number but that wasn’t the case. There was some duplicate. In order to fix this problem, I would look at it if that was intended. If there was expected behavior, I would revise my assumption. If there wasn’t meant to be, I would either look at my code if that may be the problem or not.
-
-

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

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

D. Learn and Iterate

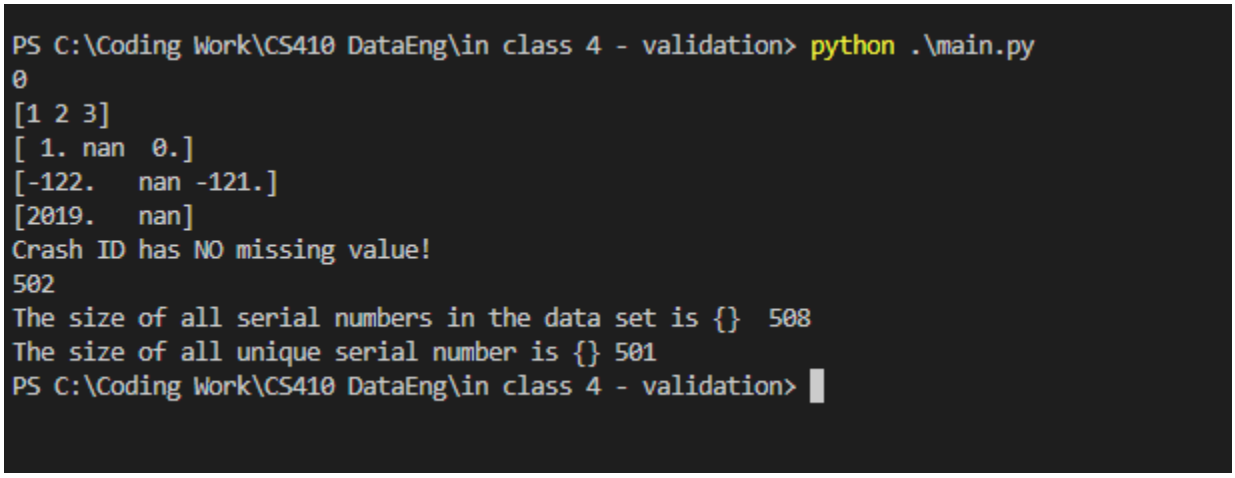
The process of validating data usually gives us a better understanding of any data set. What have you learned about the data set that you did not know at the beginning of the current ABC iteration?

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.



```python
import pandas as pd

myVar = pd.read_csv('data.csv')

#Data validation - record type exists, no null values
# Checks if there is a null value in Record Type
    # Gets the entire Record Type Column
recordType = myVar['Record Type']
    #print the number of values of nulls. ANSW --> 0 This is valid
print(recordType.isnull().sum())

#Data validation - Every record has a record type from 1 to 3
    # get the size of the record type
    #ANSW - only displays a array from 1, 2, 3 which is expected behavior
print(recordType.unique())

#Data validation - Every Data has NAN or nhs flag of 1
    #ANSW - FALSE some of the data set has a value of 0
NHSFlag = myVar['NHS Flag']
print(NHSFlag.unique())

#Data validation - Crashes occurred have approx the same longitude degrees (-122, to -121)
    #Answ - TRUE, all data set is either -122, -121 or NAN
LongitudeDegrees = myVar['Longitude Degrees']
print(LongitudeDegrees.unique())

#Data validation - All crashes occurred in the same year (2019)
    # Answ - TRUE, all data set is either 2019 or NAN
CrashYear = myVar['Crash Year']
print(CrashYear.unique())
#Data validation - Every data set has a Crash ID
    # ANSW - TRUE, all data set does contain a id, there is not NAN
CrashId = myVar['Crash ID']
CrashIdMiss = CrashId.isnull().sum()
if (CrashIdMiss>0):
    print("{} has {} missing value(s)".format('Crash ID', CrashIdMiss))
else:
    print("{} has NO missing value!".format('Crash ID'))

# Data validation - Every data set has a unique serial #
    # ANSW - FALSE, some data set may have duplicates
SerialNumber = myVar['Serial #']
SerialNumberDropNa = SerialNumber.dropna()
print(SerialNumber.unique().size)
print("The size of all serial numbers in the data set is {} ", SerialNumberDropNa.size)
print("The size of all unique serial number is {}", SerialNumberDropNa.unique().size)
```