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: Survivors of the titanic crash, dataset from kaggle. It had data related to people (gender, age, etc). There were many missing fields and values. So he had to make assumptions to make the dataset more dense and add values to it himself.

Response 2: Pokemon database, some values were and some were incorrect and you had to verify and correct them.

Response 3: The person wasn't audible enough, so couldn't really figure out what he said.

Response 4: For me, Legal report records were sent by a company then I had to import in our database, certain fields have values associated with them. If the value sent was invalid or incorrect, I would send an error message and tell the user about valid values that need to be sent. Some fields have default values associated with it. In case of invalid or blank values sent. Those fields were populated with those default values.

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.

Answer:

1. Limit assertion. “Most accidents had injury of severity 3”
2. Existence assertion. “Every accident at least had one person involved in it”
3. *intra-record* assertions. Example: “Every crash has a unique ID”
4. *limit* assertions. Example: “Sex column contains values among (1, 2, 3, 9)”
5. Create 2+ *inter-record check* assertions. Example: “Every crash record must have a serial number associated with it.”
6. *inter-record check* “The number of vehicles id’s should match the column “Total Vehicle Count”
7. Create 2+ *summary* assertions. Example: “There were thousands of crashes but not millions”
8. *summary* assertions. “There were more number of crashes that happened in the day than in the night”
9. Create 2+ *statistical distribution assertions*. Example: “crashes are evenly/uniformly distributed throughout the months of the year.”

10. *statistical distribution assertions*. Example: “Bar chart of total number of crashes grouped by month”

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:

1. Most accidents have injury severity of 5

```
In [10]: injury_severity.dropna()

Out[10]: 8       0.0
         15      1.0
         60      5.0
         61      5.0
         62      5.0
         92      0.0
         272     1.0
         587     5.0
         667     5.0
         668     5.0
         704     5.0
         818     5.0
         819     5.0
         951     0.0
         1367    5.0
         1827    0.0
         1933    5.0
         2026    0.0
         2122    5.0
         2147    7.0
         2162    0.0
         2310    0.0
         2350    0.0
         2421    1.0
         Name: Injury Severity, dtype: float64

In [13]: injury_severity = injury_severity.dropna()

In [14]: injury_severity

Out[14]: 8       0.0
         15      1.0
         60      5.0
         61      5.0
         62      5.0
         92      0.0
         272     1.0
         587     5.0
         667     5.0
         668     5.0
         704     5.0
         818     5.0
         819     5.0
         951     0.0
         1367    5.0
         1827    0.0
         1933    5.0
         2026    0.0
         2122    5.0
         2147    7.0
         2162    0.0
         2310    0.0
         2350    0.0
         2421    1.0
         Name: Injury Severity, dtype: float64

In [15]: injury_severity.mode()

Out[15]: 0    5.0
         dtype: float64
```

2.There are 1210 accidents wherein the vehicles had no occupants in it

```
dtype: float64
```

```
In [16]: vehicle_occupant_count = df['Vehicle Occupant Count']
```

```
In [17]: vehicle_occupant_count
```

```
Out[17]: 0       NaN
         1       NaN
         2       0.0
         3       NaN
         4       0.0
                 ...
         2734    NaN
         2735    NaN
         2736    0.0
         2737    NaN
         2738    0.0
         Name: Vehicle Occupant Count, Length: 2739, dtype: float64
```

```
In [18]: vehicle_occupant_count = vehicle_occupant_count.dropna()
```

```
In [19]: vehicle_occupant_count
```

```
Out[19]: 2       0.0
         4       0.0
         7       0.0
         8       0.0
         11      0.0
                 ...
         2728    0.0
         2731    0.0
         2733    0.0
         2736    0.0
         2738    0.0
         Name: Vehicle Occupant Count, Length: 1216, dtype: float64
```

```
In [23]: count = 0;
         for index, value in vehicle_occupant_count.items():
             if value <= 0:
                 count+=1

         print(count)
```

```
1210
```

3. There are duplicates but I think each record is still identified with the crash_id number which is unique to each accident. There are in total 508 crashes mentioned in the datasheet

```
In [28]: crash_ids = crash_ids.drop_duplicates(keep='first')
```

```
In [30]: crash_ids
```

```
Out[30]: 0       1809119
         5       1809229
         9       1809637
         12      1810874
         16      1812266
                  ...
         2716    1860371
         2719    1860417
         2724    1860427
         2729    1860453
         2734    1860771
         Name: Crash ID, Length: 508, dtype: int64
```

```
In [31]: len(crash_ids)
```

```
Out[31]: 508
```

4. There are some incorrect values, as there are some values which are not among (1, 2, 3, 9)

```
In [32]: sex = df['Sex']

In [34]: sex = sex.dropna()

In [37]: sex_check = sex.isin([1, 2, 3, 9])

In [42]: for index, value in sex_check.items():
             if not value:
                 print(index)
```

```
25
53
62
99
106
122
134
139
141
150
159
164
169
170
173
175
178
181
182
```

```
In [43]: sex[25]

Out[43]: 4.0
```

5. Every crash has an associated serial number.

```
In [157]: serialNumbers = CrashesDF['Serial #'].dropna().drop_duplicates()

          for index, value in UniqueCrashIds.items():
              serialNumberPerId = CrashesDF.loc[CrashesDF['Crash ID'] == value    , 'Serial #']
              if int(serialNumberPerId.tolist()[0]) == 0:
                  print("notValid")
```

```
In [ ]:
```

6. There are violations for example for crash id (1847540) there are 4 vehicle ids mentioned. But in the column ‘Total Vehicle Count’ the value is 3 (which is incorrect)

```
ParticipantsDF = ParticipantsDF.dropna(axis=1,now= all )

In [46]: VehiclesDF['Crash ID'].value_counts()

Out[46]: 1847540    4
         1812266    4
         1849023    4
         1854339    4
         1833535    4
                   ..
         1853519    1
         1844942    1
         1837049    1
         1844484    1
         1847385    1
         Name: Crash ID, Length: 508, dtype: int64

In [53]: CrashesDF

Out[53]: 0       0.0
         5       0.0
         9       0.0
         12      0.0
         16      1.0
                ...
         2716    0.0
         2719    0.0
         2724    0.0
         2729    0.0
         2734    0.0
         Name: Total Vehicle Count, Length: 508, dtype: float64

In [54]: CrashesDF.loc[CrashesDF['Crash ID'] == 1847540    , 'Total Vehicle Count']

Out[54]: 1385    3.0
         Name: Total Vehicle Count, dtype: float64

In [ ]:
```

7. There are only 508 unique crash ids, so there are not even thousands of crashes in the given dataset.

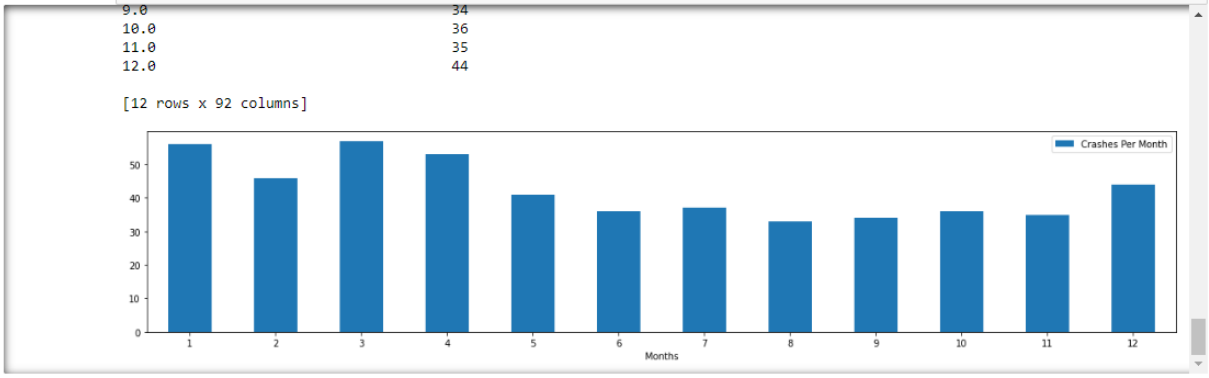
```
In [28]: crash_ids = crash_ids.drop_duplicates(keep='first')

In [30]: crash_ids

Out[30]: 0       1809119
         5       1809229
         9       1809637
         12      1810874
         16      1812266
                  ...
         2716    1860371
         2719    1860417
         2724    1860427
         2729    1860453
         2734    1860771
         Name: Crash ID, Length: 508, dtype: int64

In [31]: len(crash_ids)

Out[31]: 508
```

8. Yes there were more crashes during the day than during the night (i.e 307 in the day and 201 at night)

```
In [27]: crash_hour = df['Crash Hour']

In [28]: crash_hour = crash_hour.dropna()

In [34]: crash_hour_check = crash_hour.isin([9, 10, 11, 12, 13, 14, 15, 16, 17])

In [35]: crashDayTimeCount = 0
         crashNightTimeCount = 0
         for index, value in crash_hour_check.items():
             if value:
                 crashDayTimeCount += 1
             else:
                 crashNightTimeCount += 1

In [36]: crashDayTimeCount
Out[36]: 307

In [37]: crashNightTimeCount
Out[37]: 201

In [ ]:
```

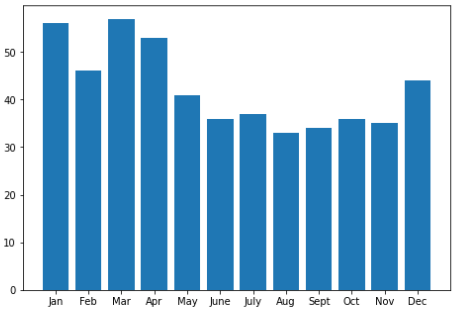
9. They do not follow a uniform distribution and certain months have more probability of crashes than others

```
In [149]: crash_per_month_stats = CrashesDF.groupby(["Crash Month"])
          crash_per_month_count = crash_per_month_stats.count()
          print(crash_per_month_count)
          YAxis = crash_per_month_count['Crash ID'].tolist()
          XAxis = ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12']

          statDf = pd.DataFrame({'Months': XAxis, 'Crashes Per Month': YAxis})
          axisDF = statDf.interpolate().plot.bar(x='Months', y='Crashes Per Month', figsize=(20, 4), rot= 0)
          9.0                                 34
          10.0                                36
          11.0                                35
          12.0                                44

          [12 rows x 92 columns]
```



10. The maximum crashes occurred in the month of March and the least crashes occurred in the month of August

```
In [38]: crash_month = df['Crash Month']

In [39]: crash_month = crash_month.dropna()

In [42]: crash_month.value_counts()

Out[42]: 3.0     57
         1.0     56
         4.0     53
         2.0     46
         12.0    44
         5.0     41
         7.0     37
         10.0    36
         6.0     36
         11.0    35
         9.0     34
         8.0     33
         Name: Crash Month, dtype: int64

In [43]: import matplotlib.pyplot as plt
         fig = plt.figure()
         ax = fig.add_axes([0,0,1,1])
         months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'June', 'July', 'Aug', 'Sept', 'Oct', 'Nov', 'Dec']
         crash_per_month_count = [56, 46, 57, 53, 41, 36, 37, 33, 34, 36, 35, 44]
         ax.bar(months,crash_per_month_count)
         plt.show()
```




```
In [ ]:
```

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.

1. Revise assumption
2. Revise assumption
3. Nothing to do, ignore
4. discard the violating row(s)
5. Nothing to do, ignore

6. Correct the values based on unique vehicle id’s mentioned
7. Nothing to do, ignore
8. Nothing to do, ignore
9. Revise assumption

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.