DataEng S24: 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.

Due: this Friday at 10pm PT
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.

A. [MUST] Initial Discussion Question

Discuss the following question among your working group members at the beginning of the week and place your own response(s) 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:

Me: Working to validate, update, and add to a GeoJSON dataset recording geographic information on Elementary, Middle, and High Schools in Washington State- I encountered a lot of erroneous or non-existent addresses. I also found that any small alteration in the lat/long coordinates could easily cause the point to be unrepresentative of the true location. This made verification through ArcGIS with the proper coordinate system tedious but very necessary.

Trae: In working with a food api database he dealt with irrelevant columns and erroneous data in those columns. He chose to solve this problem by removing the columns entirely as they did not pertain to the project.

Will: I worked with a real-estate company trying to predict a homeowner's likelihood of foreclosing in a year from their date of purchase. The data came from various brokers and therefore various formats (api, graphQL, csv dump) and parsing through the concatenating the information was hard. One broker in particular had important values where both "NaN" and "0" meant "none" while anything higher meant those values. For instance one was "number_of_credit_cards" and some values meant "NaN" (like they don't know) and some meant "0" they didn't have any cards. That was annoying. Also we had street addresses and various formats which were also annoying when trying to connect information from various brokers.

Background

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 multi-step process.

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

B. [MUST] 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.

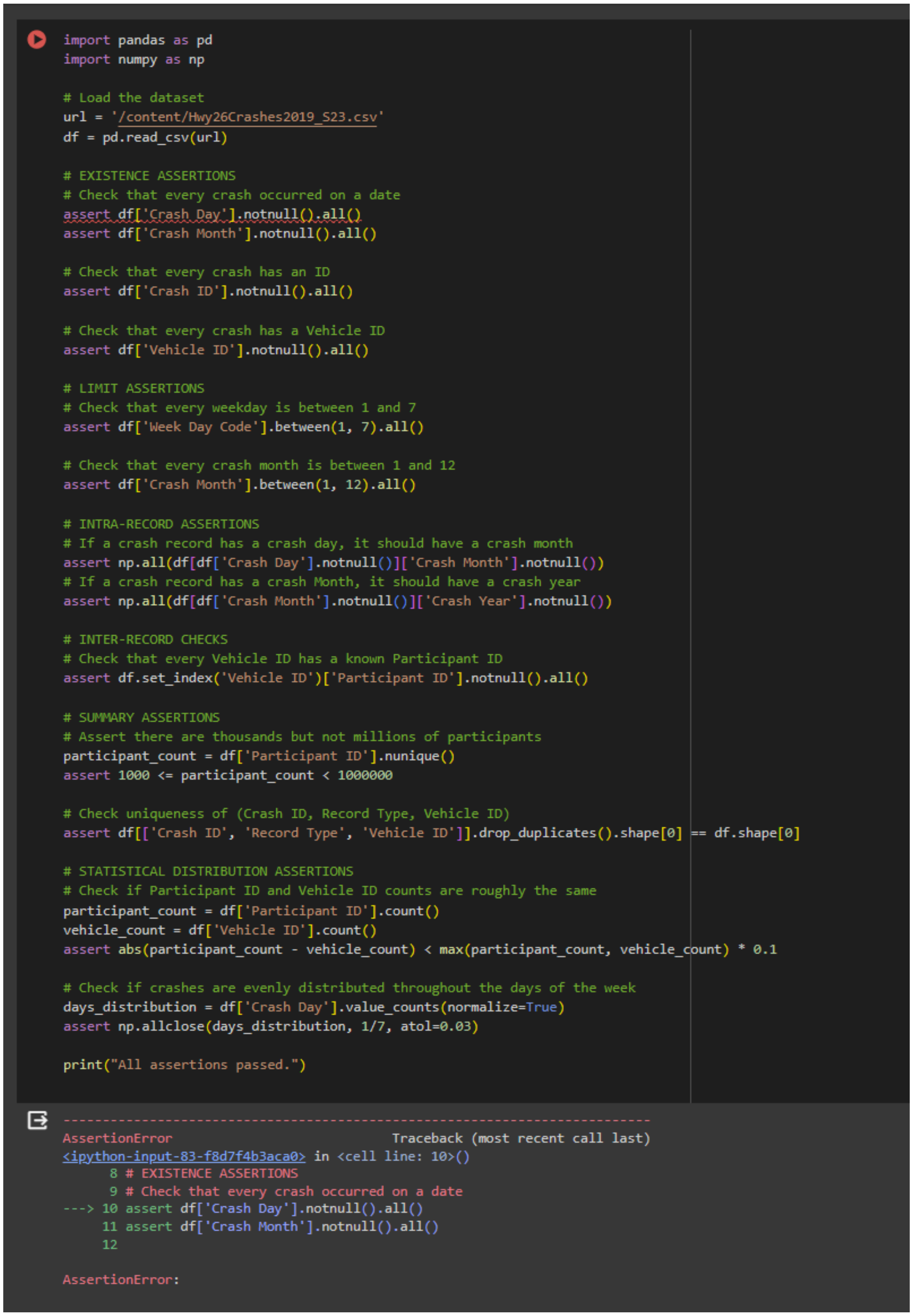
1. *existence* assertions. Example: "Every crash occurred on a date"
   a. Every Crash has an ID
   b. Every Crash has a Vehicle ID
2. *limit* assertions. Example: "Every crash occurred during year 2019"
   a. Every weekday is between 1 and 7
   b. Every crash month is between 1 and 12
3. *intra-record* assertions. Example: "If a crash record has a latitude coordinate then it should also have a longitude coordinate"
   a. If a crash record has a crash day it should have a crash month
   b. If a crash record has a crash month it should have a crash year
4. Create 2+ *inter-record check* assertions. Example: "Every vehicle listed in the crash data was part of a known crash"
   a. Every Vehicle has a known participant owner
   b. Every Crash has at least one known participant

5. Create 2+ *summary* assertions. Example: “There were thousands of crashes but not millions”
    a. There were thousands of participants not millions
    b. Every (Crash ID, Record Type, Vehicle ID) is unique
6. Create 2+ *statistical distribution assertions*. Example: “crashes are evenly/uniformly distributed throughout the months of the year.”
    a. Count for Participant ID and Vehicle ID is roughly the same
    b. Crashes are evenly distributed throughout each day of the week

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

C. [MUST] 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.



```
import pandas as pd
import numpy as np

# Load the dataset
url = '/content/Hwy26Crashes2019_S23.csv'
df = pd.read_csv(url)

# EXISTENCE ASSERTIONS
# Check that every crash occurred on a date
assert df['Crash Day'].notnull().all()
assert df['Crash Month'].notnull().all()

# Check that every crash has an ID
assert df['Crash ID'].notnull().all()

# Check that every crash has a Vehicle ID
assert df['Vehicle ID'].notnull().all()

# LIMIT ASSERTIONS
# Check that every weekday is between 1 and 7
assert df['Week Day Code'].between(1, 7).all()

# Check that every crash month is between 1 and 12
assert df['Crash Month'].between(1, 12).all()

# INTRA-RECORD ASSERTIONS
# If a crash record has a crash day, it should have a crash month
assert np.all(df[df['Crash Day'].notnull()]['Crash Month'].notnull())
# If a crash record has a crash Month, it should have a crash year
assert np.all(df[df['Crash Month'].notnull()]['Crash Year'].notnull())

# INTER-RECORD CHECKS
# Check that every Vehicle ID has a known Participant ID
assert df.set_index('Vehicle ID')['Participant ID'].notnull().all()

# SUMMARY ASSERTIONS
# Assert there are thousands but not millions of participants
participant_count = df['Participant ID'].nunique()
assert 1000 <= participant_count < 1000000

# Check uniqueness of (Crash ID, Record Type, Vehicle ID)
assert df[['Crash ID', 'Record Type', 'Vehicle ID']].drop_duplicates().shape[0] == df.shape[0]

# STATISTICAL DISTRIBUTION ASSERTIONS
# Check if Participant ID and Vehicle ID counts are roughly the same
participant_count = df['Participant ID'].count()
vehicle_count = df['Vehicle ID'].count()
assert abs(participant_count - vehicle_count) < max(participant_count, vehicle_count) * 0.1

# Check if crashes are evenly distributed throughout the days of the week
days_distribution = df['Crash Day'].value_counts(normalize=True)
assert np.allclose(days_distribution, 1/7, atol=0.03)

print("All assertions passed.")
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
<ipython-input-83-f8d7f4b3aca0> in <cell line: 10>()
      8 # EXISTENCE ASSERTIONS
      9 # Check that every crash occurred on a date
---> 10 assert df['Crash Day'].notnull().all()
     11 assert df['Crash Month'].notnull().all()
     12 

AssertionError:
```

D. [MUST] Run Your Code and Analyze the Results

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

- Some records do not have a crash day or month recorded
- With no crash day or month they cannot be between the limit
- Without a crash day we cannot check that it also has a crash month
- There were duplicate records for ('Crash ID', 'Record Type', 'Vehicle ID')
- Participant count is actually much larger than Vehicle count so this check should be changed

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. For each assertion related to Crash Date- the change should be assertion revision. Upon further review, each record has a record type and only records of type 1 contain the date info so the assertion should pertain to type 1 records.
2. For the Duplicate records violation- the change should again be assertion revision. There can be duplicate records with ('Crash ID', 'Record Type', 'Vehicle ID') if there were multiple participants with that vehicle. So the change needs to be to add Participant ID.
3. The participant Count and Vehicle count should not be equal; this was a misinterpretation of the dataset. This assertion should be ignored and a new assertion for this type should be considered.

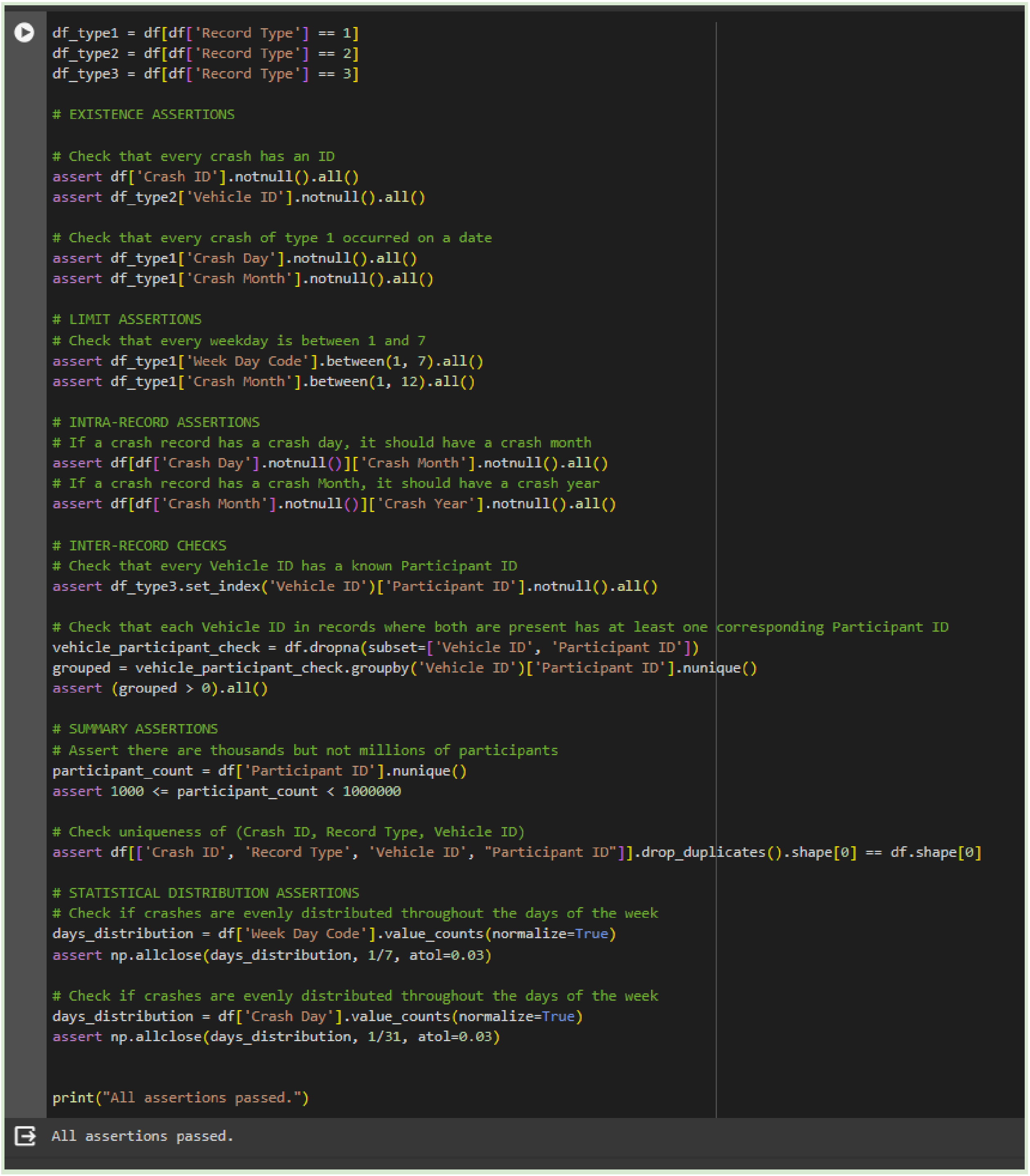
E. [SHOULD] Resolve the Violations and Transform the Data

For each assertion violation write python code to resolve the violation according to your entry in the “how to resolve” section above.

A. Create New Assertions

a. Every crash record type 1 occurred on a date
b. Every crash record type 1 occurred on a weekday is between 1 and 7
c. Every crash record type 1 crash month is between 1 and 12
d. Every crash record type 1 with a crash days should have a crash month
e. Every crash record type 1 with a crash month should have a crash year
f. Every crash record type 3 with a vehicle ID has a Participant ID
g. Every crash record (Crash ID, Record Type, Vehicle ID, Participant ID) is unique
h. Every Crash record with a Vehicle ID has at least 1 Participant ID
i. Crashes are evenly distributed throughout each day of the month
j. Every crash record type 2 has a vehicle ID

B. Validate / Code



```python
df_type1 = df[df['Record Type'] == 1]
df_type2 = df[df['Record Type'] == 2]
df_type3 = df[df['Record Type'] == 3]

# EXISTENCE ASSERTIONS

# Check that every crash has an ID
assert df['Crash ID'].notnull().all()
assert df_type2['Vehicle ID'].notnull().all()

# Check that every crash of type 1 occurred on a date
assert df_type1['Crash Day'].notnull().all()
assert df_type1['Crash Month'].notnull().all()

# LIMIT ASSERTIONS
# Check that every weekday is between 1 and 7
assert df_type1['Week Day Code'].between(1, 7).all()
assert df_type1['Crash Month'].between(1, 12).all()

# INTRA-RECORD ASSERTIONS
# If a crash record has a crash day, it should have a crash month
assert df[df['Crash Day'].notnull()]['Crash Month'].notnull().all()
# If a crash record has a crash Month, it should have a crash year
assert df[df['Crash Month'].notnull()]['Crash Year'].notnull().all()

# INTER-RECORD CHECKS
# Check that every Vehicle ID has a known Participant ID
assert df_type3.set_index('Vehicle ID')['Participant ID'].notnull().all()

# Check that each Vehicle ID in records where both are present has at least one corresponding Participant ID
vehicle_participant_check = df.dropna(subset=['Vehicle ID', 'Participant ID'])
grouped = vehicle_participant_check.groupby('Vehicle ID')['Participant ID'].nunique()
assert (grouped > 0).all()

# SUMMARY ASSERTIONS
# Assert there are thousands but not millions of participants
participant_count = df['Participant ID'].nunique()
assert 1000 <= participant_count < 1000000

# Check uniqueness of (Crash ID, Record Type, Vehicle ID)
assert df[['Crash ID', 'Record Type', 'Vehicle ID', "Participant ID"]].drop_duplicates().shape[0] == df.shape[0]

# STATISTICAL DISTRIBUTION ASSERTIONS
# Check if crashes are evenly distributed throughout the days of the week
days_distribution = df['Week Day Code'].value_counts(normalize=True)
assert np.allclose(days_distribution, 1/7, atol=0.03)

# Check if crashes are evenly distributed throughout the days of the week
days_distribution = df['Crash Day'].value_counts(normalize=True)
assert np.allclose(days_distribution, 1/31, atol=0.03)


print("All assertions passed.")
```

```
All assertions passed.
```

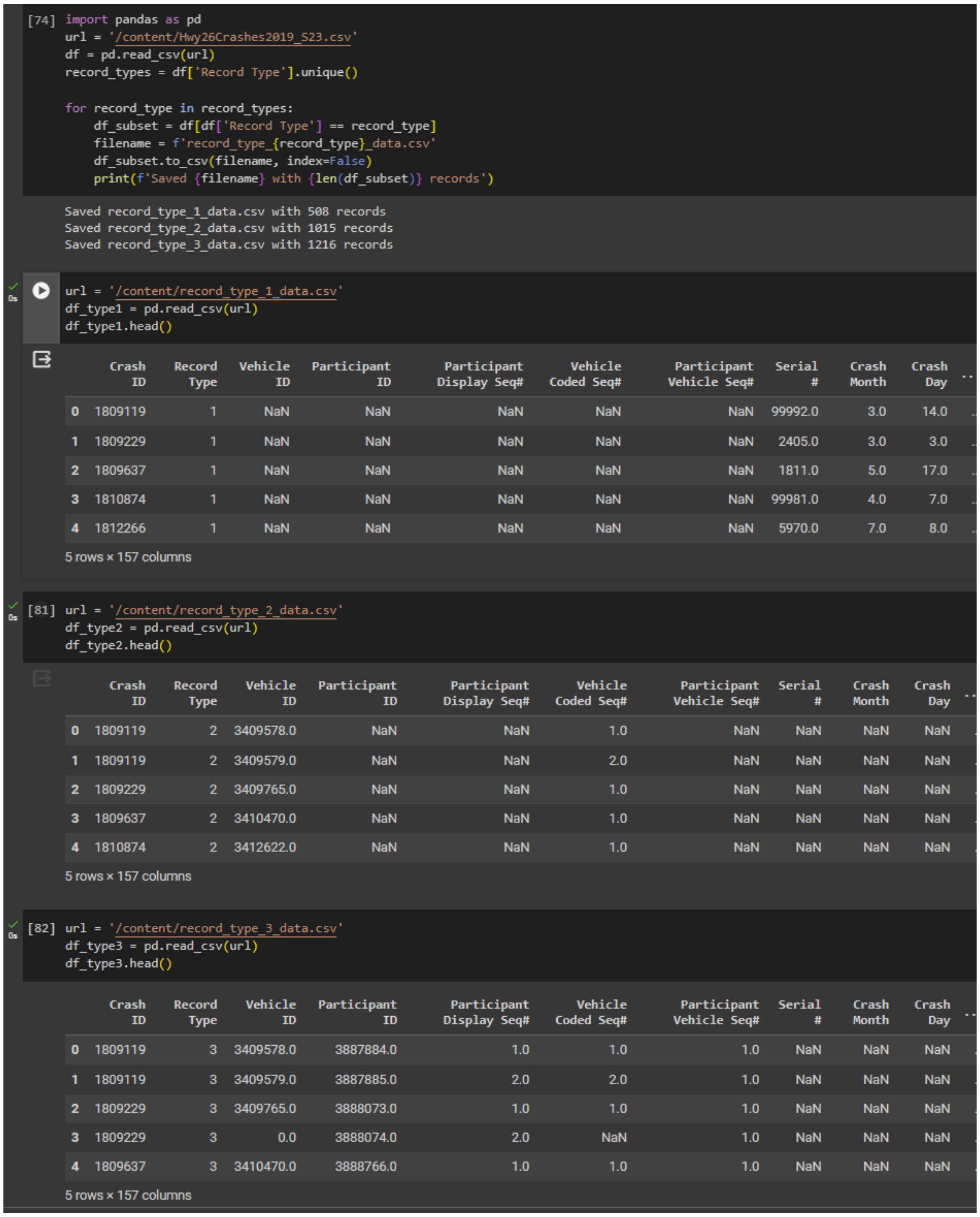
C. Analyze Results

a. All assertions pass!

b. However I did need to specify the record type. To me this seems as though these should be 3 separate tables that can be joined together if need be.

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.

For this section I will address the fact that I had to specify record type for my validation assertions. I will separate the data into 3 different files based on their record type.



```
[74] import pandas as pd
     url = '/content/Hwy26Crashes2019_S23.csv'
     df = pd.read_csv(url)
     record_types = df['Record Type'].unique()

     for record_type in record_types:
         df_subset = df[df['Record Type'] == record_type]
         filename = f'record_type_{record_type}_data.csv'
         df_subset.to_csv(filename, index=False)
         print(f'Saved {filename} with {len(df_subset)} records')
```

```
Saved record_type_1_data.csv with 508 records
Saved record_type_2_data.csv with 1015 records
Saved record_type_3_data.csv with 1216 records
```

```
url = '/content/record_type_1_data.csv'
df_type1 = pd.read_csv(url)
df_type1.head()
```

	Crash ID	Record Type	Vehicle ID	Participant ID	Participant Display Seq#	Vehicle Coded Seq#	Participant Vehicle Seq#	Serial #	Crash Month	Crash Day	…
0	1809119	1	NaN	NaN	NaN	NaN	NaN	99992.0	3.0	14.0	…
1	1809229	1	NaN	NaN	NaN	NaN	NaN	2405.0	3.0	3.0	…
2	1809637	1	NaN	NaN	NaN	NaN	NaN	1811.0	5.0	17.0	…
3	1810874	1	NaN	NaN	NaN	NaN	NaN	99981.0	4.0	7.0	…
4	1812266	1	NaN	NaN	NaN	NaN	NaN	5970.0	7.0	8.0	…

5 rows × 157 columns

```
[81] url = '/content/record_type_2_data.csv'
     df_type2 = pd.read_csv(url)
     df_type2.head()
```

	Crash ID	Record Type	Vehicle ID	Participant ID	Participant Display Seq#	Vehicle Coded Seq#	Participant Vehicle Seq#	Serial #	Crash Month	Crash Day	…
0	1809119	2	3409578.0	NaN	NaN	1.0	NaN	NaN	NaN	NaN	…
1	1809119	2	3409579.0	NaN	NaN	2.0	NaN	NaN	NaN	NaN	…
2	1809229	2	3409765.0	NaN	NaN	1.0	NaN	NaN	NaN	NaN	…
3	1809637	2	3410470.0	NaN	NaN	1.0	NaN	NaN	NaN	NaN	…
4	1810874	2	3412622.0	NaN	NaN	1.0	NaN	NaN	NaN	NaN	…

5 rows × 157 columns

```
[82] url = '/content/record_type_3_data.csv'
     df_type3 = pd.read_csv(url)
     df_type3.head()
```

	Crash ID	Record Type	Vehicle ID	Participant ID	Participant Display Seq#	Vehicle Coded Seq#	Participant Vehicle Seq#	Serial #	Crash Month	Crash Day	…
0	1809119	3	3409578.0	3887884.0	1.0	1.0	1.0	NaN	NaN	NaN	…
1	1809119	3	3409579.0	3887885.0	2.0	2.0	1.0	NaN	NaN	NaN	…
2	1809229	3	3409765.0	3888073.0	1.0	1.0	1.0	NaN	NaN	NaN	…
3	1809229	3	0.0	3888074.0	2.0	NaN	1.0	NaN	NaN	NaN	…
4	1809637	3	3410470.0	3888766.0	1.0	1.0	1.0	NaN	NaN	NaN	…

5 rows × 157 columns

These are essentially 3 files about Crashes, Vehicles, and Participants respectively as you can see which IDs are NAN in each file.

F. [ASPIRE] 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 B, C, D and E at least one more time.