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:

I have worked with data that had errors. The errors were obtained because of the null values in the dataset. The data was expected to be in string format, but some were integers. I fixed the data type and used it.

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

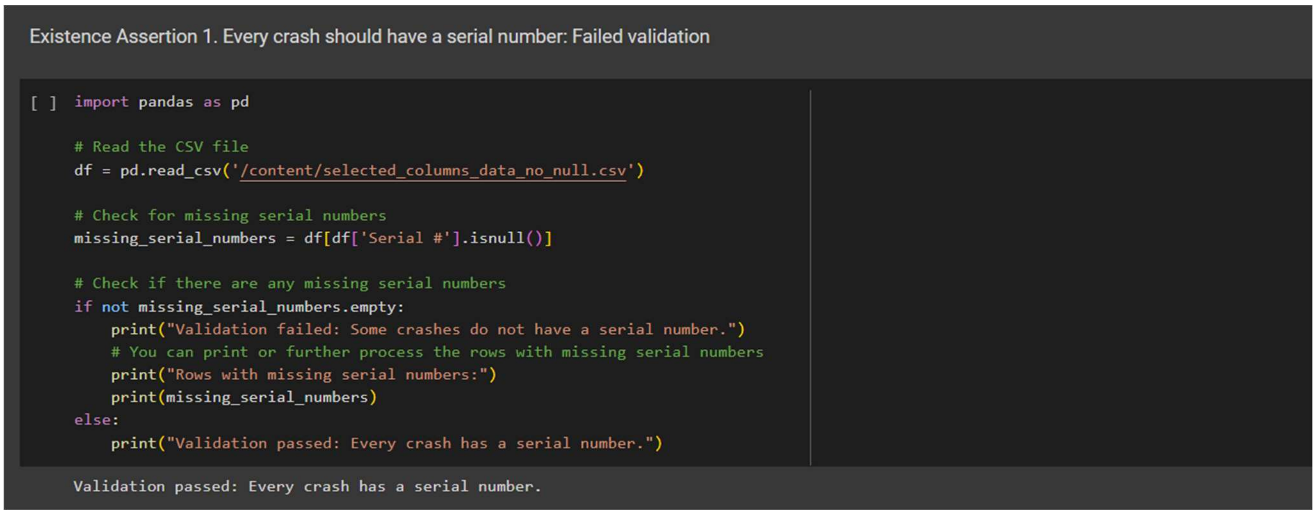
Data validation is usually an iterative multi-step process.

B. Create assertions about the data
B. Write code to evaluate your assertions.
B. Run the code, analyze the results
B. 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.

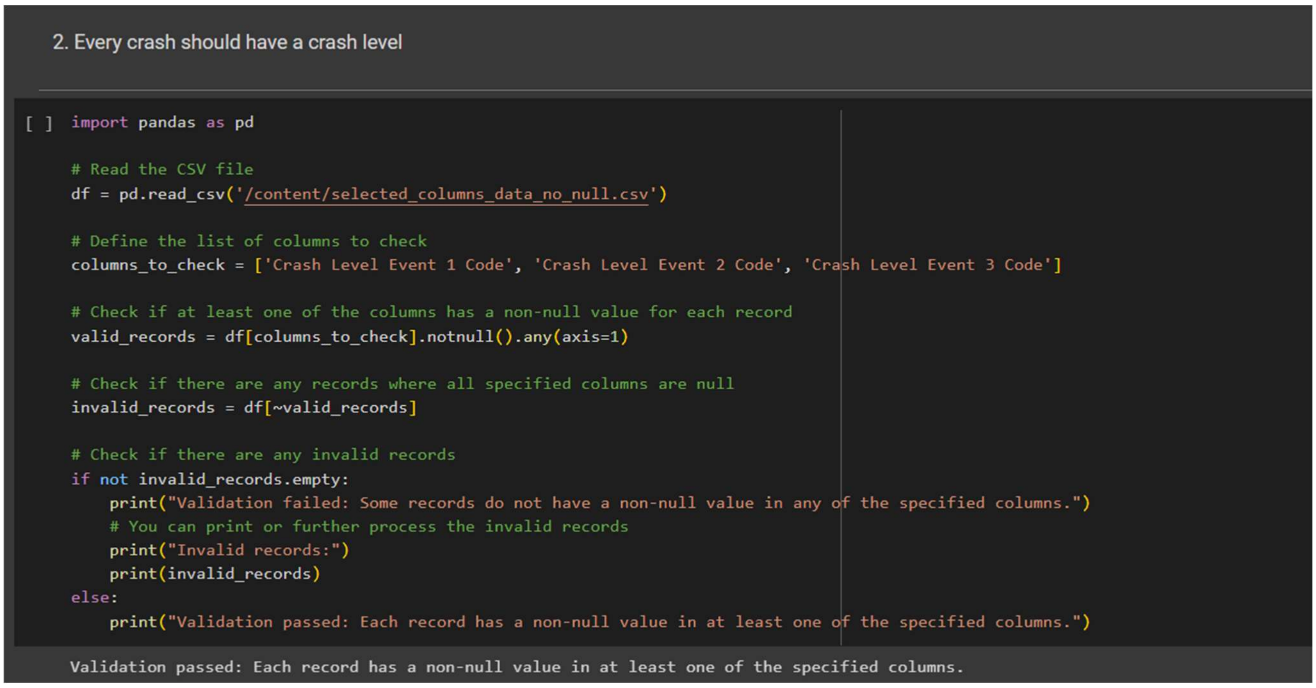
Existence assertion : Example: “Every crash occurred on a date”
Every crash should have a serial number: failed validation



Existence Assertion 1. Every crash should have a serial number: Failed validation

```
[ ] import pandas as pd

    # Read the CSV file
    df = pd.read_csv('/content/selected_columns_data_no_null.csv')

    # Check for missing serial numbers
    missing_serial_numbers = df[df['Serial #'].isnull()]

    # Check if there are any missing serial numbers
    if not missing_serial_numbers.empty:
        print("Validation failed: Some crashes do not have a serial number.")
        # You can print or further process the rows with missing serial numbers
        print("Rows with missing serial numbers:")
        print(missing_serial_numbers)
    else:
        print("Validation passed: Every crash has a serial number.")

    Validation passed: Every crash has a serial number.
```

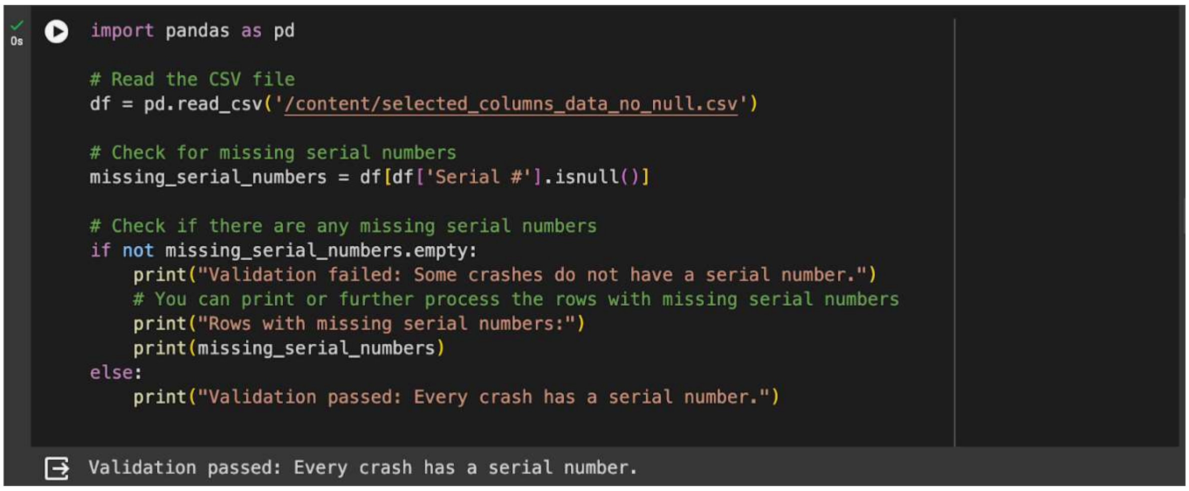
Every crash should have a crash level.: failed validation



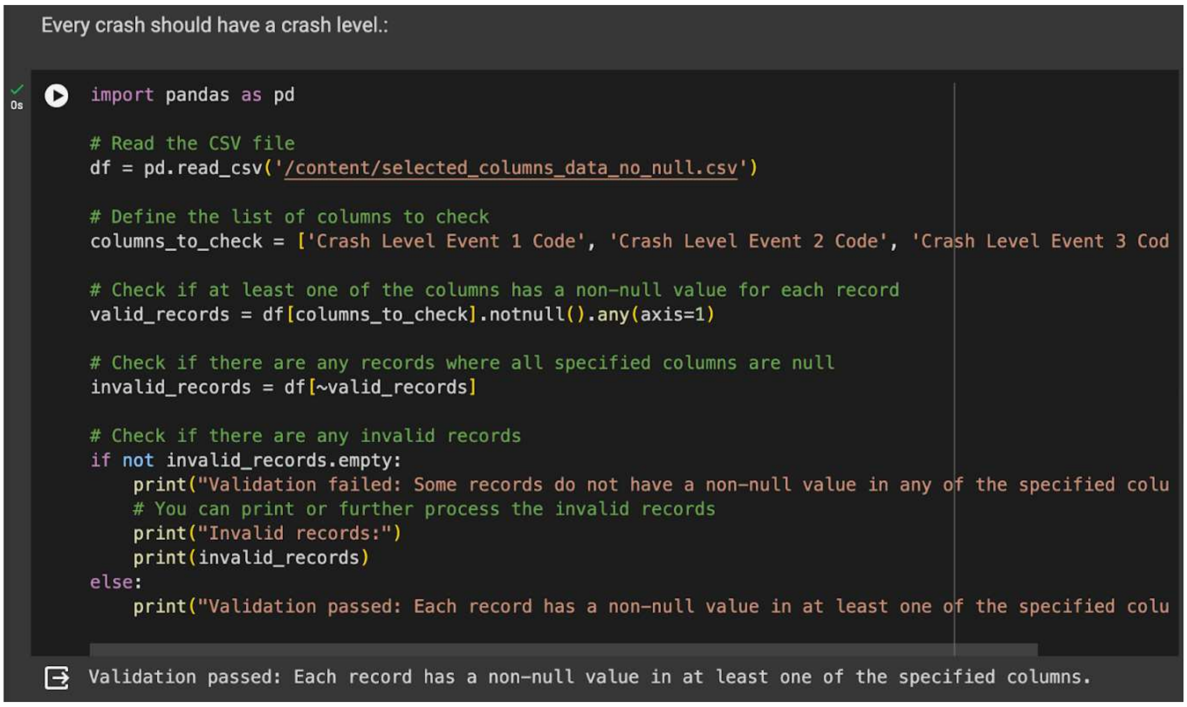
2. Every crash should have a crash level

```
[ ] import pandas as pd

    # Read the CSV file
    df = pd.read_csv('/content/selected_columns_data_no_null.csv')

    # Define the list of columns to check
    columns_to_check = ['Crash Level Event 1 Code', 'Crash Level Event 2 Code', 'Crash Level Event 3 Code']

    # Check if at least one of the columns has a non-null value for each record
    valid_records = df[columns_to_check].notnull().any(axis=1)

    # Check if there are any records where all specified columns are null
    invalid_records = df[~valid_records]

    # Check if there are any invalid records
    if not invalid_records.empty:
        print("Validation failed: Some records do not have a non-null value in any of the specified columns.")
        # You can print or further process the invalid records
        print("Invalid records:")
        print(invalid_records)
    else:
        print("Validation passed: Each record has a non-null value in at least one of the specified columns.")

    Validation passed: Each record has a non-null value in at least one of the specified columns.
```

Validation passed



```
0s  import pandas as pd

    # Read the CSV file
    df = pd.read_csv('/content/selected_columns_data_no_null.csv')

    # Check for missing serial numbers
    missing_serial_numbers = df[df['Serial #'].isnull()]

    # Check if there are any missing serial numbers
    if not missing_serial_numbers.empty:
        print("Validation failed: Some crashes do not have a serial number.")
        # You can print or further process the rows with missing serial numbers
        print("Rows with missing serial numbers:")
        print(missing_serial_numbers)
    else:
        print("Validation passed: Every crash has a serial number.")

    Validation passed: Every crash has a serial number.
```

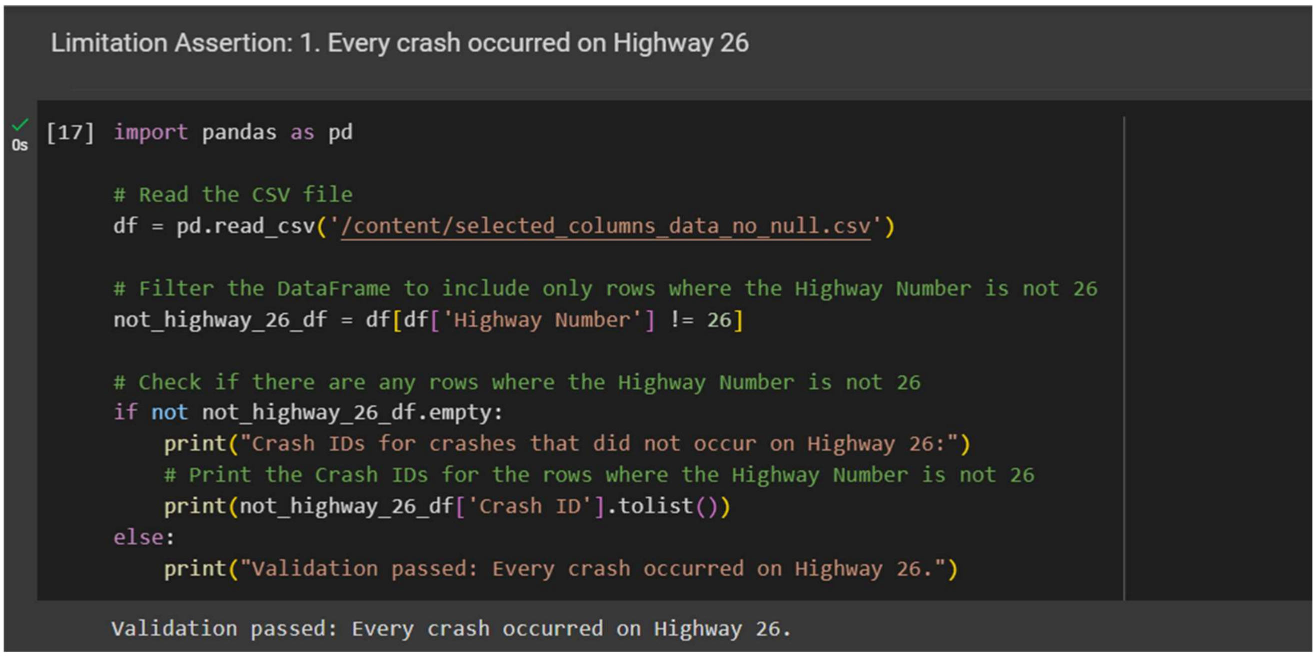

Every crash should have a crash level.:

```
import pandas as pd

# Read the CSV file
df = pd.read_csv('/content/selected_columns_data_no_null.csv')

# Define the list of columns to check
columns_to_check = ['Crash Level Event 1 Code', 'Crash Level Event 2 Code', 'Crash Level Event 3 Cod

# Check if at least one of the columns has a non-null value for each record
valid_records = df[columns_to_check].notnull().any(axis=1)

# Check if there are any records where all specified columns are null
invalid_records = df[~valid_records]

# Check if there are any invalid records
if not invalid_records.empty:
    print("Validation failed: Some records do not have a non-null value in any of the specified colu
    # You can print or further process the invalid records
    print("Invalid records:")
    print(invalid_records)
else:
    print("Validation passed: Each record has a non-null value in at least one of the specified colu
```

```
Validation passed: Each record has a non-null value in at least one of the specified columns.
```

Limit assertion : Example: “Every crash occurred during the year 2019”

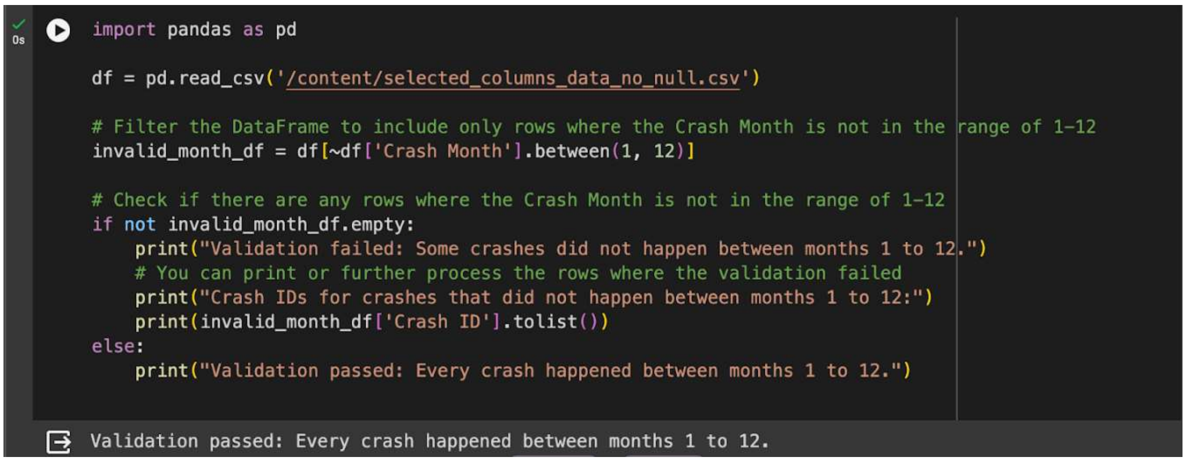
Every crash occurred on Highway 26: validation passed


Limitation Assertion: 1. Every crash occurred on Highway 26

```
[17] import pandas as pd

     # Read the CSV file
     df = pd.read_csv('/content/selected_columns_data_no_null.csv')

     # Filter the DataFrame to include only rows where the Highway Number is not 26
     not_highway_26_df = df[df['Highway Number'] != 26]

     # Check if there are any rows where the Highway Number is not 26
     if not not_highway_26_df.empty:
         print("Crash IDs for crashes that did not occur on Highway 26:")
         # Print the Crash IDs for the rows where the Highway Number is not 26
         print(not_highway_26_df['Crash ID'].tolist())
     else:
         print("Validation passed: Every crash occurred on Highway 26.")
```

```
Validation passed: Every crash occurred on Highway 26.
```

Every crash should have happened between month 1-12 : validation passed

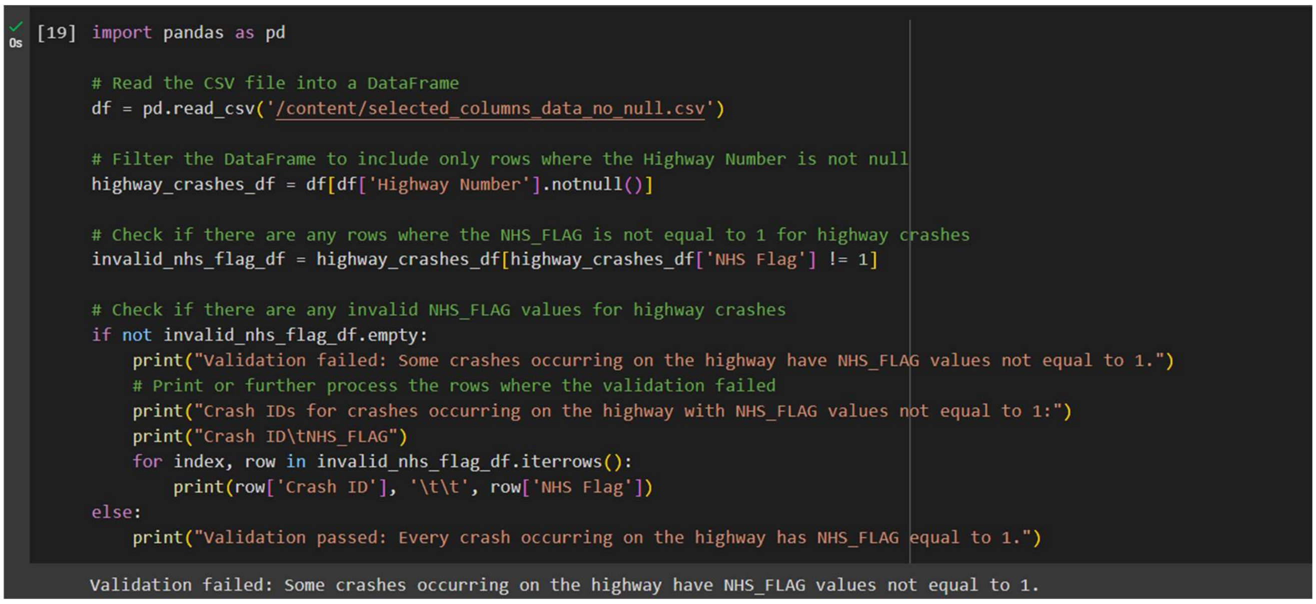


```
import pandas as pd

df = pd.read_csv('/content/selected_columns_data_no_null.csv')

# Filter the DataFrame to include only rows where the Crash Month is not in the range of 1-12
invalid_month_df = df[~df['Crash Month'].between(1, 12)]

# Check if there are any rows where the Crash Month is not in the range of 1-12
if not invalid_month_df.empty:
    print("Validation failed: Some crashes did not happen between months 1 to 12.")
    # You can print or further process the rows where the validation failed
    print("Crash IDs for crashes that did not happen between months 1 to 12:")
    print(invalid_month_df['Crash ID'].tolist())
else:
    print("Validation passed: Every crash happened between months 1 to 12.")
```

```
Validation passed: Every crash happened between months 1 to 12.
```

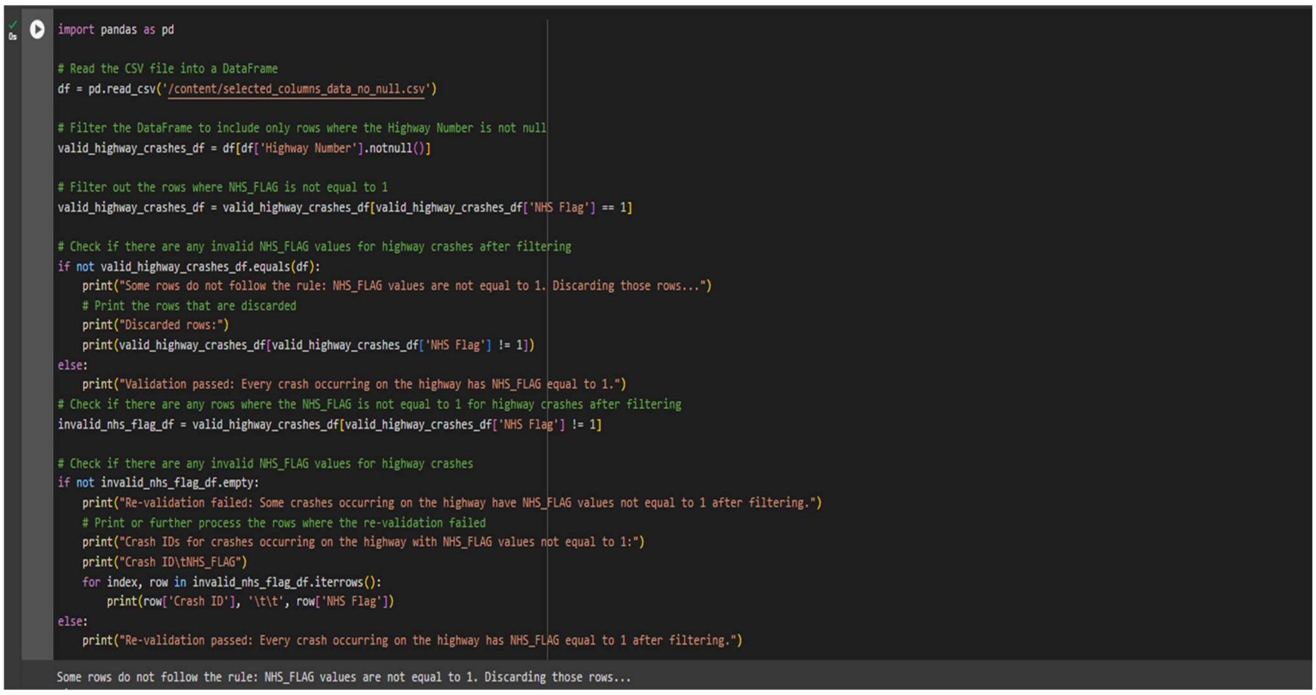
Intra-record assertion: Example: “If a crash record has a latitude coordinate then it should also have a longitude coordinate”

For every crash that occurs on the Highway, NHS_FLAG should be 1.
Validation failed: Some crashes occurring on the highway have NHS_FLAG values not equal to 1.
Crash IDs for crashes occurring on the highway with NHS_FLAG values not equal to 1:
[1834202, 1834440, 1834858, 1839856, 1840680, 1841331, 1841506, 1843445, 1843709, 1845603, 1847023, 1847352, 1847898, 1848047, 1849337, 1849597, 1851564, 1854013, 1854398, 1854987, 1856784, 1857340, 1858723, 1859565, 1860043]



```
[19] import pandas as pd

     # Read the CSV file into a DataFrame
     df = pd.read_csv('/content/selected_columns_data_no_null.csv')

     # Filter the DataFrame to include only rows where the Highway Number is not null
     highway_crashes_df = df[df['Highway Number'].notnull()]

     # Check if there are any rows where the NHS_FLAG is not equal to 1 for highway crashes
     invalid_nhs_flag_df = highway_crashes_df[highway_crashes_df['NHS Flag'] != 1]

     # Check if there are any invalid NHS_FLAG values for highway crashes
     if not invalid_nhs_flag_df.empty:
         print("Validation failed: Some crashes occurring on the highway have NHS_FLAG values not equal to 1.")
         # Print or further process the rows where the validation failed
         print("Crash IDs for crashes occurring on the highway with NHS_FLAG values not equal to 1:")
         print("Crash ID\tNHS_FLAG")
         for index, row in invalid_nhs_flag_df.iterrows():
             print(row['Crash ID'], '\t\t', row['NHS Flag'])
     else:
         print("Validation passed: Every crash occurring on the highway has NHS_FLAG equal to 1.")
```

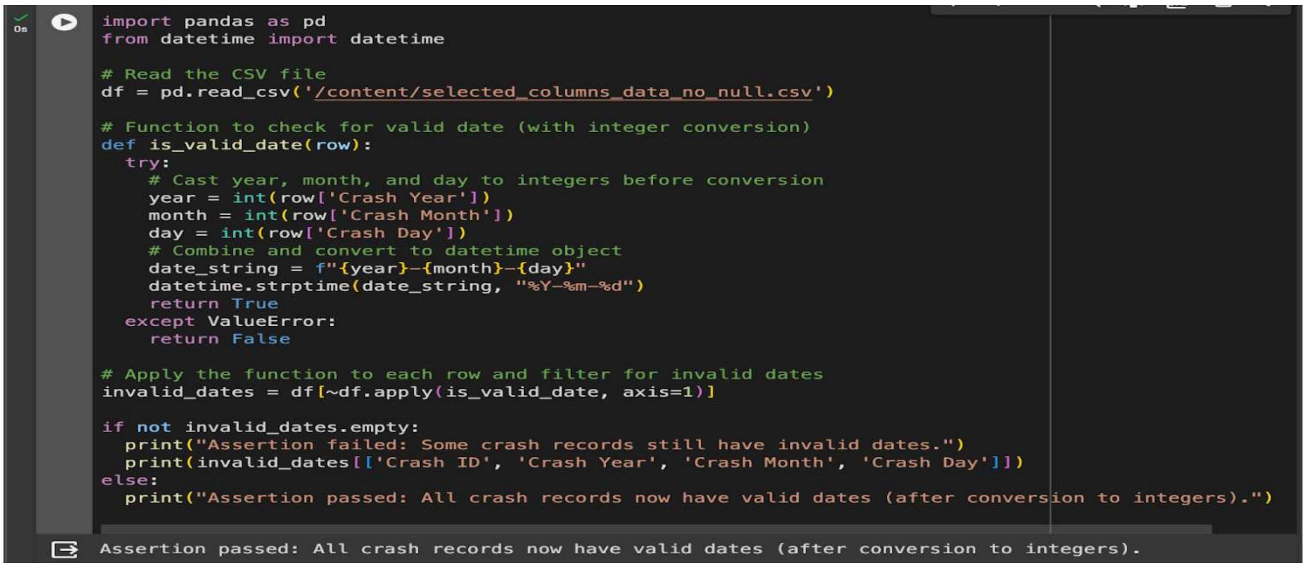
```
Validation failed: Some crashes occurring on the highway have NHS_FLAG values not equal to 1.
```



```python
import pandas as pd

# Read the CSV file into a DataFrame
df = pd.read_csv('/content/selected_columns_data_no_null.csv')

# Filter the DataFrame to include only rows where the Highway Number is not null
valid_highway_crashes_df = df[df['Highway Number'].notnull()]

# Filter out the rows where NHS_FLAG is not equal to 1
valid_highway_crashes_df = valid_highway_crashes_df[valid_highway_crashes_df['NHS Flag'] == 1]

# Check if there are any invalid NHS_FLAG values for highway crashes after filtering
if not valid_highway_crashes_df.equals(df):
    print("Some rows do not follow the rule: NHS_FLAG values are not equal to 1. Discarding those rows...")
    # Print the rows that are discarded
    print("Discarded rows:")
    print(valid_highway_crashes_df[valid_highway_crashes_df['NHS Flag'] != 1])
else:
    print("Validation passed: Every crash occurring on the highway has NHS_FLAG equal to 1.")
# Check if there are any rows where the NHS_FLAG is not equal to 1 for highway crashes after filtering
invalid_nhs_flag_df = valid_highway_crashes_df[valid_highway_crashes_df['NHS Flag'] != 1]

# Check if there are any invalid NHS_FLAG values for highway crashes
if not invalid_nhs_flag_df.empty:
    print("Re-validation failed: Some crashes occurring on the highway have NHS_FLAG values not equal to 1 after filtering.")
    # Print or further process the rows where the re-validation failed
    print("Crash IDs for crashes occurring on the highway with NHS_FLAG values not equal to 1:")
    print("Crash ID\tNHS_FLAG")
    for index, row in invalid_nhs_flag_df.iterrows():
        print(row['Crash ID'], '\t\t', row['NHS Flag'])
else:
    print("Re-validation passed: Every crash occurring on the highway has NHS_FLAG equal to 1 after filtering.")
```

```
Some rows do not follow the rule: NHS_FLAG values are not equal to 1. Discarding those rows...
```

The values in 'Crash Year', 'Crash Month', and 'Crash Day' columns should represent a valid date : Validation passes

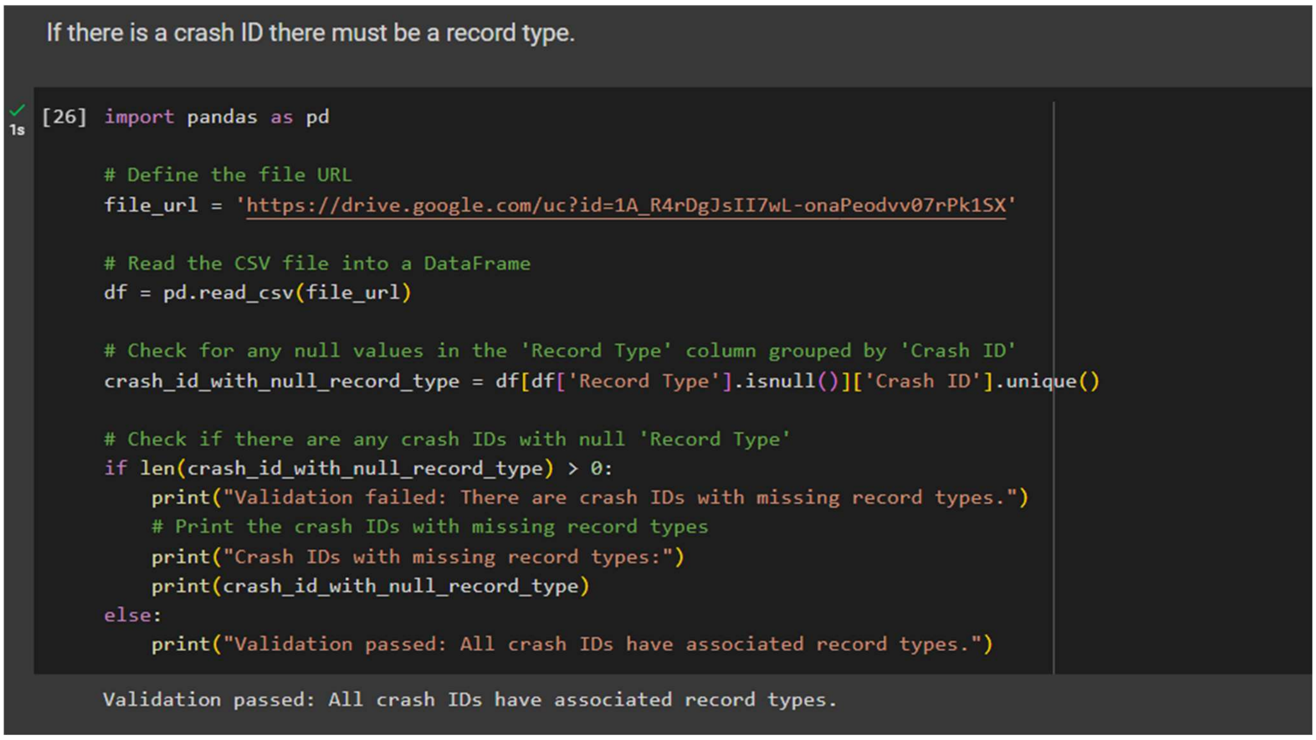


```python
import pandas as pd
from datetime import datetime

# Read the CSV file
df = pd.read_csv('/content/selected_columns_data_no_null.csv')

# Function to check for valid date (with integer conversion)
def is_valid_date(row):
  try:
    # Cast year, month, and day to integers before conversion
    year = int(row['Crash Year'])
    month = int(row['Crash Month'])
    day = int(row['Crash Day'])
    # Combine and convert to datetime object
    date_string = f"{year}-{month}-{day}"
    datetime.strptime(date_string, "%Y-%m-%d")
    return True
  except ValueError:
    return False

# Apply the function to each row and filter for invalid dates
invalid_dates = df[~df.apply(is_valid_date, axis=1)]

if not invalid_dates.empty:
  print("Assertion failed: Some crash records still have invalid dates.")
  print(invalid_dates[['Crash ID', 'Crash Year', 'Crash Month', 'Crash Day']])
else:
  print("Assertion passed: All crash records now have valid dates (after conversion to integers).")
```

```
Assertion passed: All crash records now have valid dates (after conversion to integers).
```

Inter-record check assertion:
Example: “Every vehicle listed in the crash data was part of a known crash”

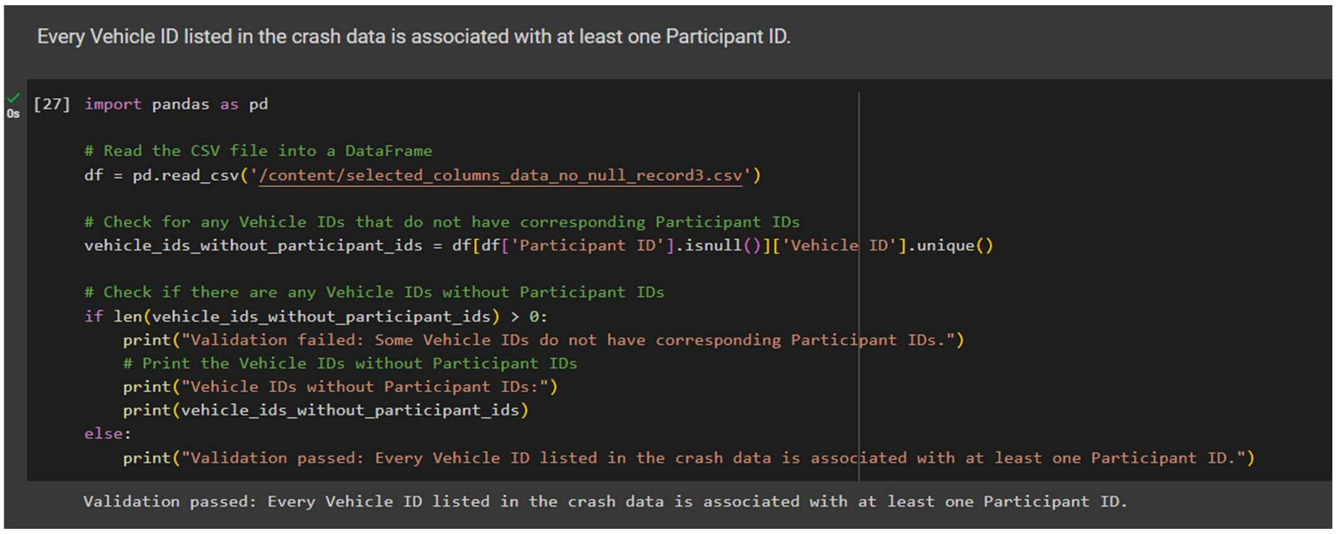
If there is a crash ID there must be a record type.
Validation passed: For every crash ID, there is a record type.


If there is a crash ID there must be a record type.

```
[26] import pandas as pd

     # Define the file URL
     file_url = 'https://drive.google.com/uc?id=1A_R4rDgJsII7wL-onaPeodvv07rPk1SX'

     # Read the CSV file into a DataFrame
     df = pd.read_csv(file_url)

     # Check for any null values in the 'Record Type' column grouped by 'Crash ID'
     crash_id_with_null_record_type = df[df['Record Type'].isnull()]['Crash ID'].unique()

     # Check if there are any crash IDs with null 'Record Type'
     if len(crash_id_with_null_record_type) > 0:
         print("Validation failed: There are crash IDs with missing record types.")
         # Print the crash IDs with missing record types
         print("Crash IDs with missing record types:")
         print(crash_id_with_null_record_type)
     else:
         print("Validation passed: All crash IDs have associated record types.")
```

```
Validation passed: All crash IDs have associated record types.
```

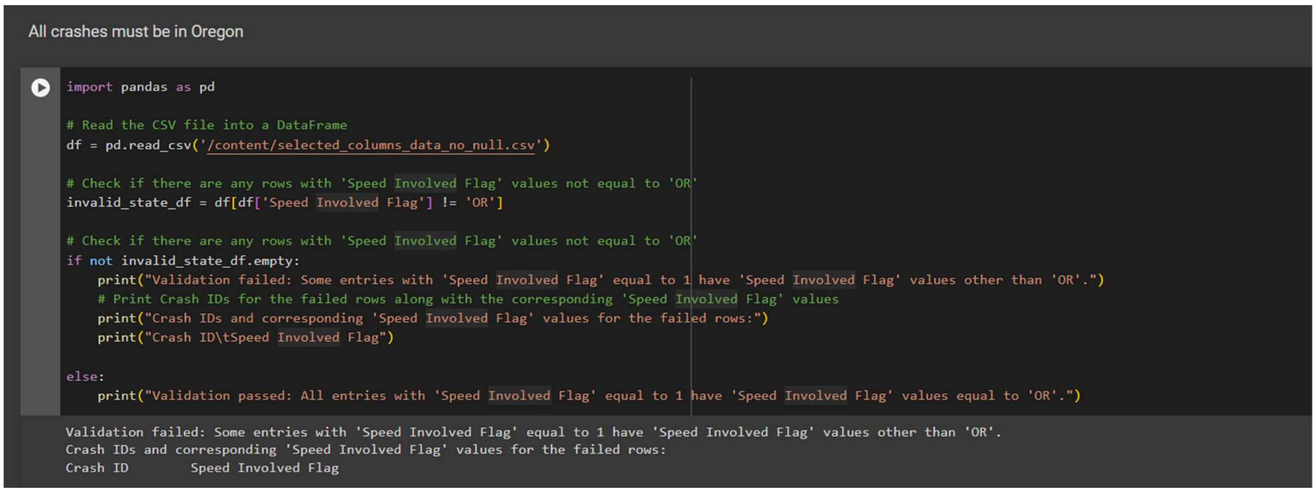
Every Vehicle ID listed in the crash data is associated with at least one Participant ID.


Every Vehicle ID listed in the crash data is associated with at least one Participant ID.

```
[27] import pandas as pd

     # Read the CSV file into a DataFrame
     df = pd.read_csv('/content/selected_columns_data_no_null_record3.csv')

     # Check for any Vehicle IDs that do not have corresponding Participant IDs
     vehicle_ids_without_participant_ids = df[df['Participant ID'].isnull()]['Vehicle ID'].unique()

     # Check if there are any Vehicle IDs without Participant IDs
     if len(vehicle_ids_without_participant_ids) > 0:
         print("Validation failed: Some Vehicle IDs do not have corresponding Participant IDs.")
         # Print the Vehicle IDs without Participant IDs
         print("Vehicle IDs without Participant IDs:")
         print(vehicle_ids_without_participant_ids)
     else:
         print("Validation passed: Every Vehicle ID listed in the crash data is associated with at least one Participant ID.")
```

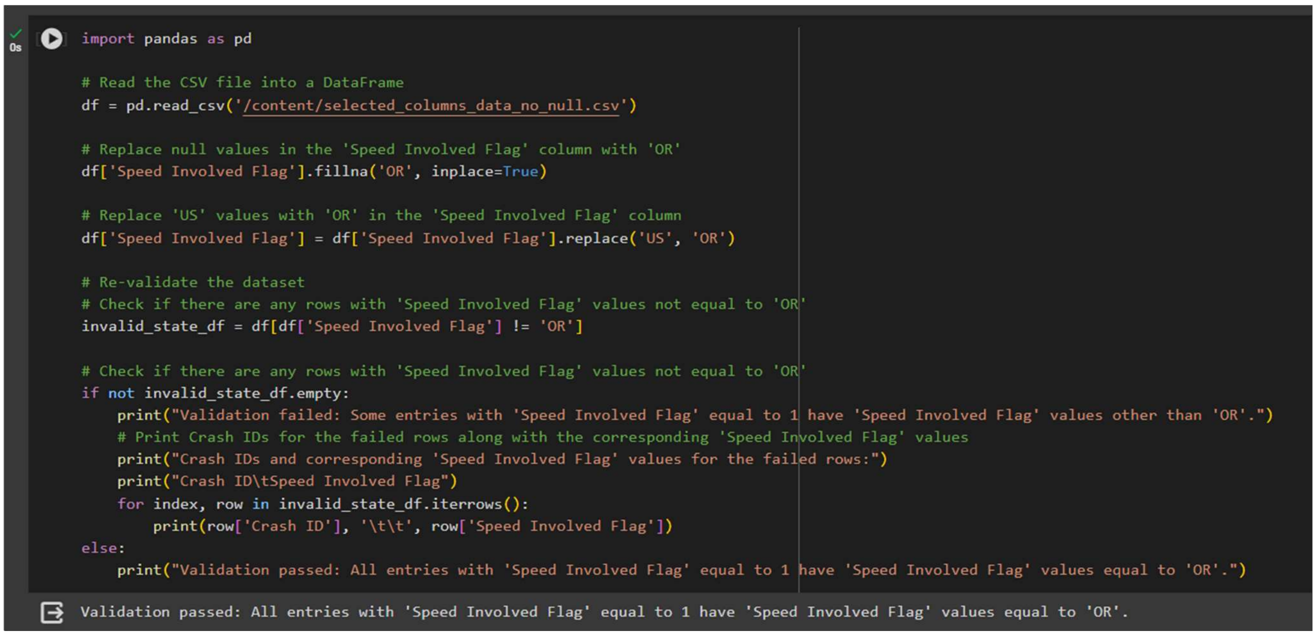
```
Validation passed: Every Vehicle ID listed in the crash data is associated with at least one Participant ID.
```

Summary assertion: Example: "There were thousands of crashes but not millions". All crashes should occur in Oregon


All crashes must be in Oregon

```
import pandas as pd

# Read the CSV file into a DataFrame
df = pd.read_csv('/content/selected_columns_data_no_null.csv')

# Check if there are any rows with 'Speed Involved Flag' values not equal to 'OR'
invalid_state_df = df[df['Speed Involved Flag'] != 'OR']

# Check if there are any rows with 'Speed Involved Flag' values not equal to 'OR'
if not invalid_state_df.empty:
    print("Validation failed: Some entries with 'Speed Involved Flag' equal to 1 have 'Speed Involved Flag' values other than 'OR'.")
    # Print Crash IDs for the failed rows along with the corresponding 'Speed Involved Flag' values
    print("Crash IDs and corresponding 'Speed Involved Flag' values for the failed rows:")
    print("Crash ID\tSpeed Involved Flag")

else:
    print("Validation passed: All entries with 'Speed Involved Flag' equal to 1 have 'Speed Involved Flag' values equal to 'OR'.")
```

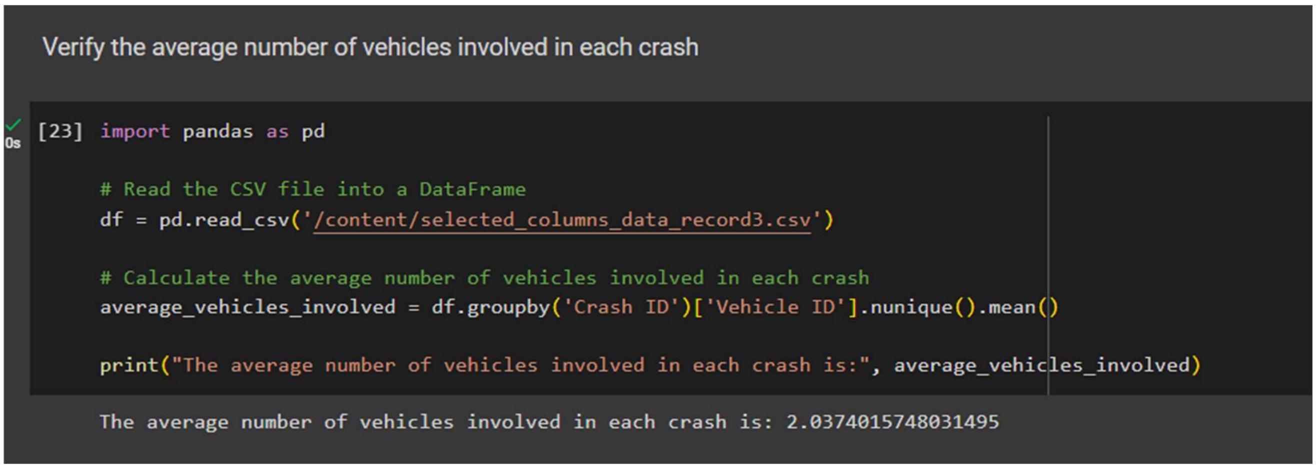
```
Validation failed: Some entries with 'Speed Involved Flag' equal to 1 have 'Speed Involved Flag' values other than 'OR'.
Crash IDs and corresponding 'Speed Involved Flag' values for the failed rows:
Crash ID        Speed Involved Flag
```



```
import pandas as pd

# Read the CSV file into a DataFrame
df = pd.read_csv('/content/selected_columns_data_no_null.csv')

# Replace null values in the 'Speed Involved Flag' column with 'OR'
df['Speed Involved Flag'].fillna('OR', inplace=True)

# Replace 'US' values with 'OR' in the 'Speed Involved Flag' column
df['Speed Involved Flag'] = df['Speed Involved Flag'].replace('US', 'OR')

# Re-validate the dataset
# Check if there are any rows with 'Speed Involved Flag' values not equal to 'OR'
invalid_state_df = df[df['Speed Involved Flag'] != 'OR']

# Check if there are any rows with 'Speed Involved Flag' values not equal to 'OR'
if not invalid_state_df.empty:
    print("Validation failed: Some entries with 'Speed Involved Flag' equal to 1 have 'Speed Involved Flag' values other than 'OR'.")
    # Print Crash IDs for the failed rows along with the corresponding 'Speed Involved Flag' values
    print("Crash IDs and corresponding 'Speed Involved Flag' values for the failed rows:")
    print("Crash ID\tSpeed Involved Flag")
    for index, row in invalid_state_df.iterrows():
        print(row['Crash ID'], '\t\t', row['Speed Involved Flag'])
else:
    print("Validation passed: All entries with 'Speed Involved Flag' equal to 1 have 'Speed Involved Flag' values equal to 'OR'.")
```

```
Validation passed: All entries with 'Speed Involved Flag' equal to 1 have 'Speed Involved Flag' values equal to 'OR'.
```

Verify the average number of vehicles involved in each crash

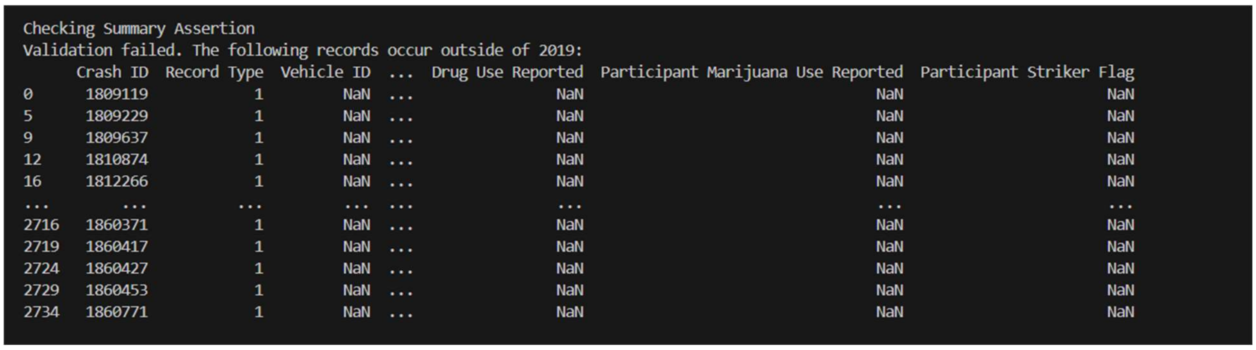


Verify the average number of vehicles involved in each crash

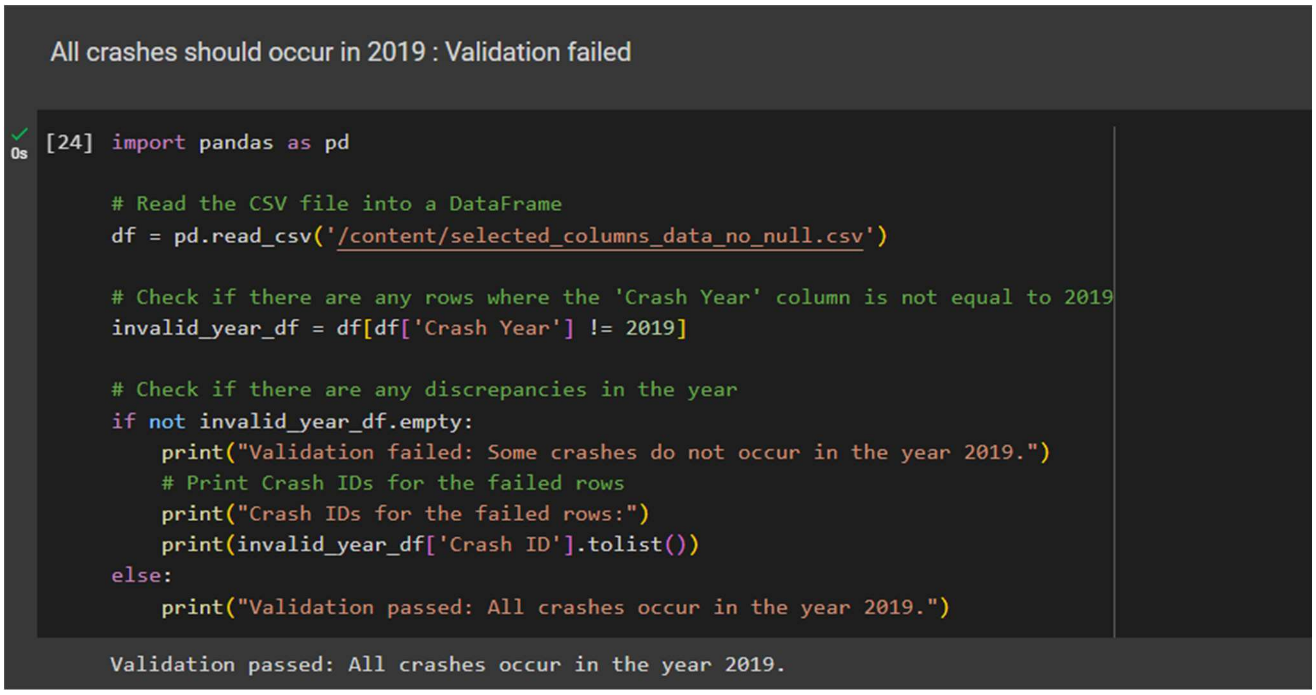
```
[23] import pandas as pd

     # Read the CSV file into a DataFrame
     df = pd.read_csv('/content/selected_columns_data_record3.csv')

     # Calculate the average number of vehicles involved in each crash
     average_vehicles_involved = df.groupby('Crash ID')['Vehicle ID'].nunique().mean()

     print("The average number of vehicles involved in each crash is:", average_vehicles_involved)
```

```
The average number of vehicles involved in each crash is: 2.0374015748031495
```

All crashes should occur in 2019 : Validation failed



```
Checking Summary Assertion
Validation failed. The following records occur outside of 2019:
      Crash ID  Record Type  Vehicle ID  ...  Drug Use Reported  Participant Marijuana Use Reported  Participant Striker Flag
0      1809119            1         NaN  ...                NaN                                 NaN                       NaN
5      1809229            1         NaN  ...                NaN                                 NaN                       NaN
9      1809637            1         NaN  ...                NaN                                 NaN                       NaN
12     1810874            1         NaN  ...                NaN                                 NaN                       NaN
16     1812266            1         NaN  ...                NaN                                 NaN                       NaN
...        ...          ...         ...  ...                ...                                 ...                       ...
2716   1860371            1         NaN  ...                NaN                                 NaN                       NaN
2719   1860417            1         NaN  ...                NaN                                 NaN                       NaN
2724   1860427            1         NaN  ...                NaN                                 NaN                       NaN
2729   1860453            1         NaN  ...                NaN                                 NaN                       NaN
2734   1860771            1         NaN  ...                NaN                                 NaN                       NaN
```

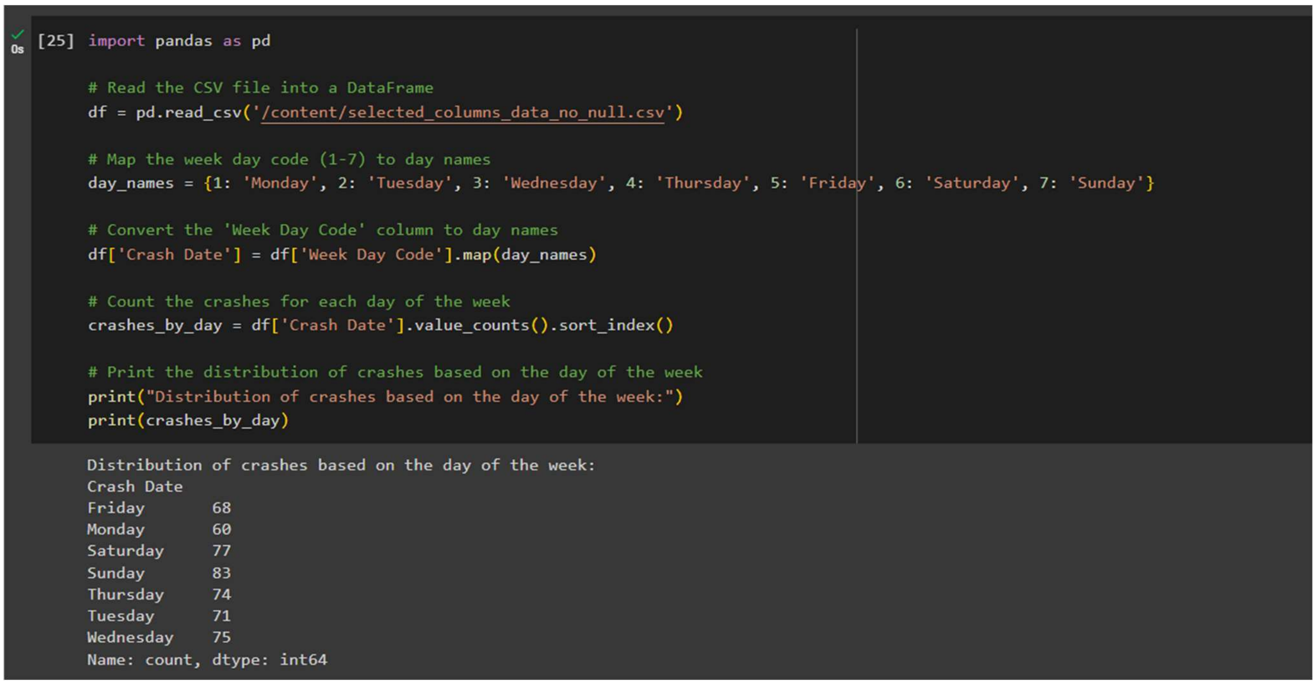

All crashes should occur in 2019 : Validation failed

```
[24] import pandas as pd

     # Read the CSV file into a DataFrame
     df = pd.read_csv('/content/selected_columns_data_no_null.csv')

     # Check if there are any rows where the 'Crash Year' column is not equal to 2019
     invalid_year_df = df[df['Crash Year'] != 2019]

     # Check if there are any discrepancies in the year
     if not invalid_year_df.empty:
         print("Validation failed: Some crashes do not occur in the year 2019.")
         # Print Crash IDs for the failed rows
         print("Crash IDs for the failed rows:")
         print(invalid_year_df['Crash ID'].tolist())
     else:
         print("Validation passed: All crashes occur in the year 2019.")
```

```
Validation passed: All crashes occur in the year 2019.
```

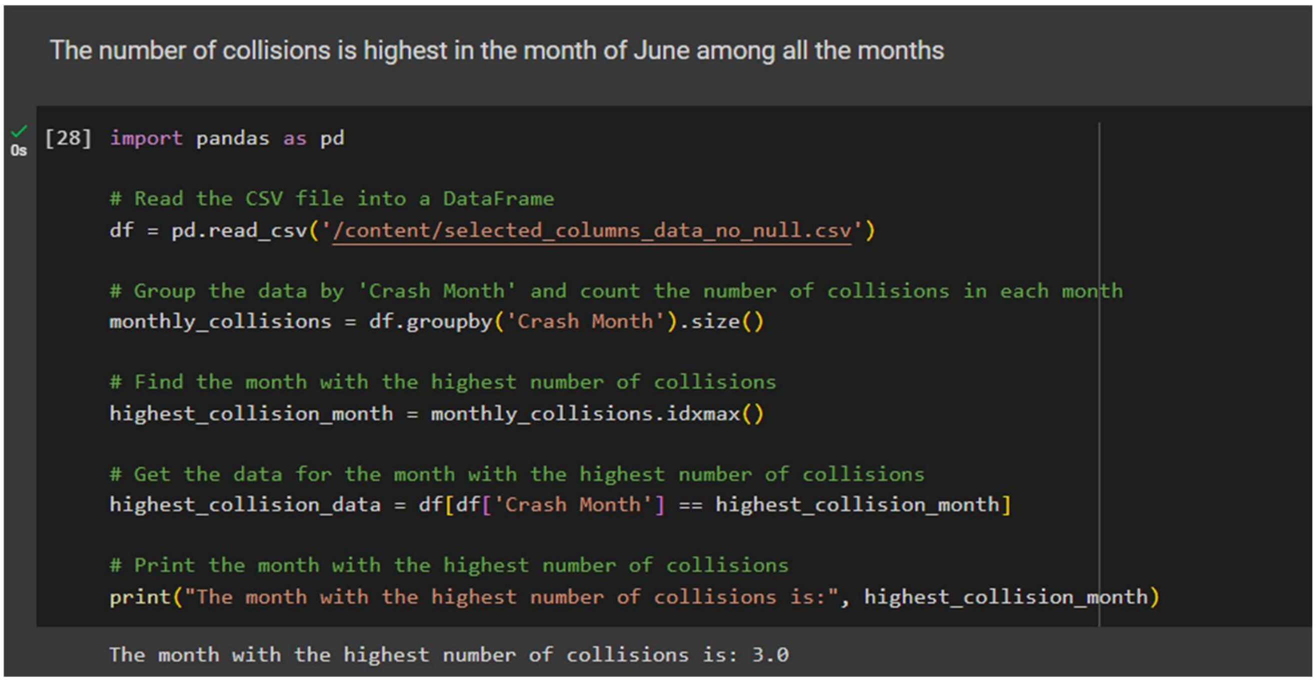
Statistical distribution assertions: Example: “crashes are evenly/uniformly distributed throughout the months of the year.”

Distribution of the crash based on the day (which day from Monday-Sunday):


```
[25] import pandas as pd

     # Read the CSV file into a DataFrame
     df = pd.read_csv('/content/selected_columns_data_no_null.csv')

     # Map the week day code (1-7) to day names
     day_names = {1: 'Monday', 2: 'Tuesday', 3: 'Wednesday', 4: 'Thursday', 5: 'Friday', 6: 'Saturday', 7: 'Sunday'}

     # Convert the 'Week Day Code' column to day names
     df['Crash Date'] = df['Week Day Code'].map(day_names)

     # Count the crashes for each day of the week
     crashes_by_day = df['Crash Date'].value_counts().sort_index()

     # Print the distribution of crashes based on the day of the week
     print("Distribution of crashes based on the day of the week:")
     print(crashes_by_day)
```

```
Distribution of crashes based on the day of the week:
Crash Date
Friday       68
Monday       60
Saturday     77
Sunday       83
Thursday     74
Tuesday      71
Wednesday    75
Name: count, dtype: int64
```

The number of collisions is highest in the months of June among all the months
This is month 3, which is march


The number of collisions is highest in the month of June among all the months

```
[28] import pandas as pd

     # Read the CSV file into a DataFrame
     df = pd.read_csv('/content/selected_columns_data_no_null.csv')

     # Group the data by 'Crash Month' and count the number of collisions in each month
     monthly_collisions = df.groupby('Crash Month').size()

     # Find the month with the highest number of collisions
     highest_collision_month = monthly_collisions.idxmax()

     # Get the data for the month with the highest number of collisions
     highest_collision_data = df[df['Crash Month'] == highest_collision_month]

     # Print the month with the highest number of collisions
     print("The month with the highest number of collisions is:", highest_collision_month)

     The month with the highest number of collisions is: 3.0
```

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.

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

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

- Some crashes do not have a serial number.
  Solution: Made use of three different csv file based on record type
- Some crashes occurring on the highway have NHS_FLAG values not equal to 1.
  Solution: As our data represents accident happening in NH 26 so will discard the rows not following this rule
- Some entries with 'Speed Involved Flag' equal to 1 have 'State' column values other than 'OR'.
  Solution: We will replace US with oregon as the field that doesn't have OR has US as their value, for null values we just added OR.

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.

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.

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.
Ans: Validated everything under B

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?