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:

Mahesh Vankayalapati: Yes, I have worked on suicide data, were I have collected the raw data from police station records and worked on them like fixing the missing values, deleting the duplicates and restructuring the data set for applying different Algorithms in RapidMiner.

Addagalla Lakshmi Sai Prasanna: While working on the DBMS project, I used data from kaggle where there were few rows that had missing data, junk characters.

Venkata Bhuvana Kancharla : I have worked with data of users whose accounts were created in an investment company. I have found some errors like null values and duplicate users.

Neha Khawle: Yes, while working on an AI/ML course project, the data we had was not clean and had missing values. And did preprocessing on those data.

Manisha: I have worked with some data with errors, it was like some of the data were not in required types. We were expecting data in form of a string but it was integer. Later changed it ans 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
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 should have a serial number:** failed validation
    b. **Every crash should have a crash level**.: failed validation

Solution: made three different csv file based on record type

```python
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.")
```

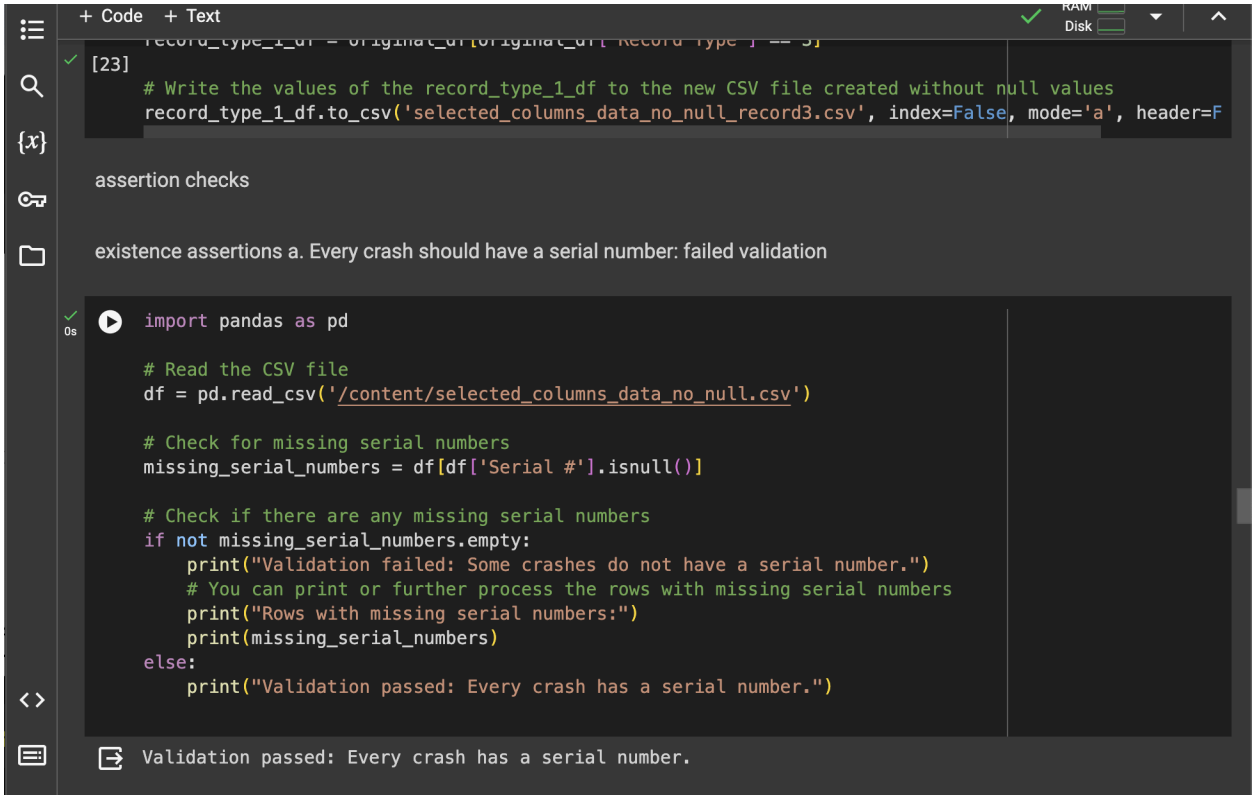
```
Validation passed: Every crash has a serial number.
```

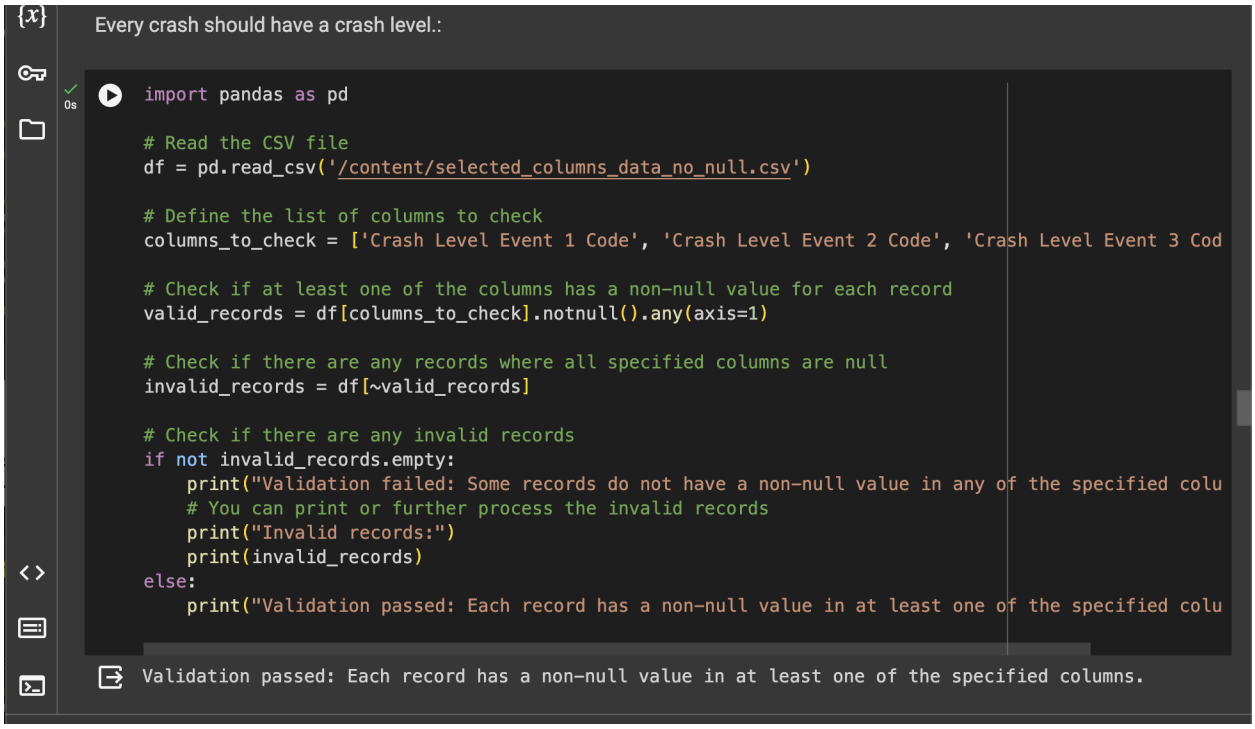
+ Code + Text

```python
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', 'Cra

# 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 o
    # 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 o
```

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

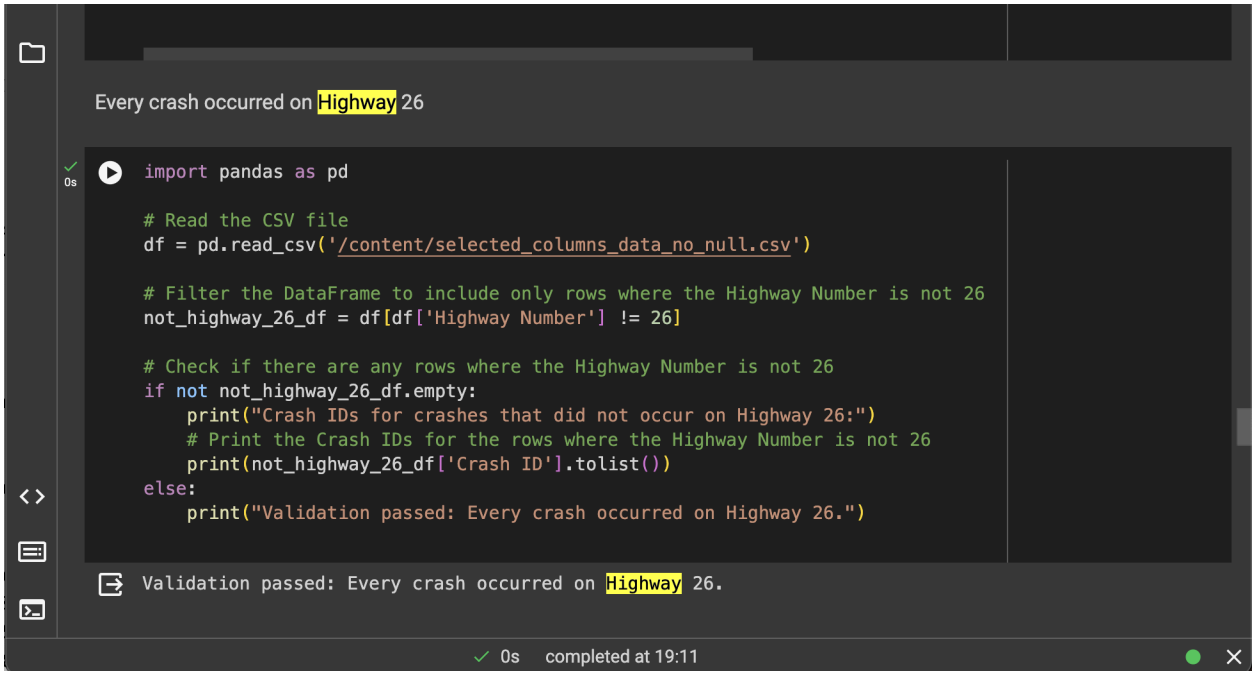
Validation passing



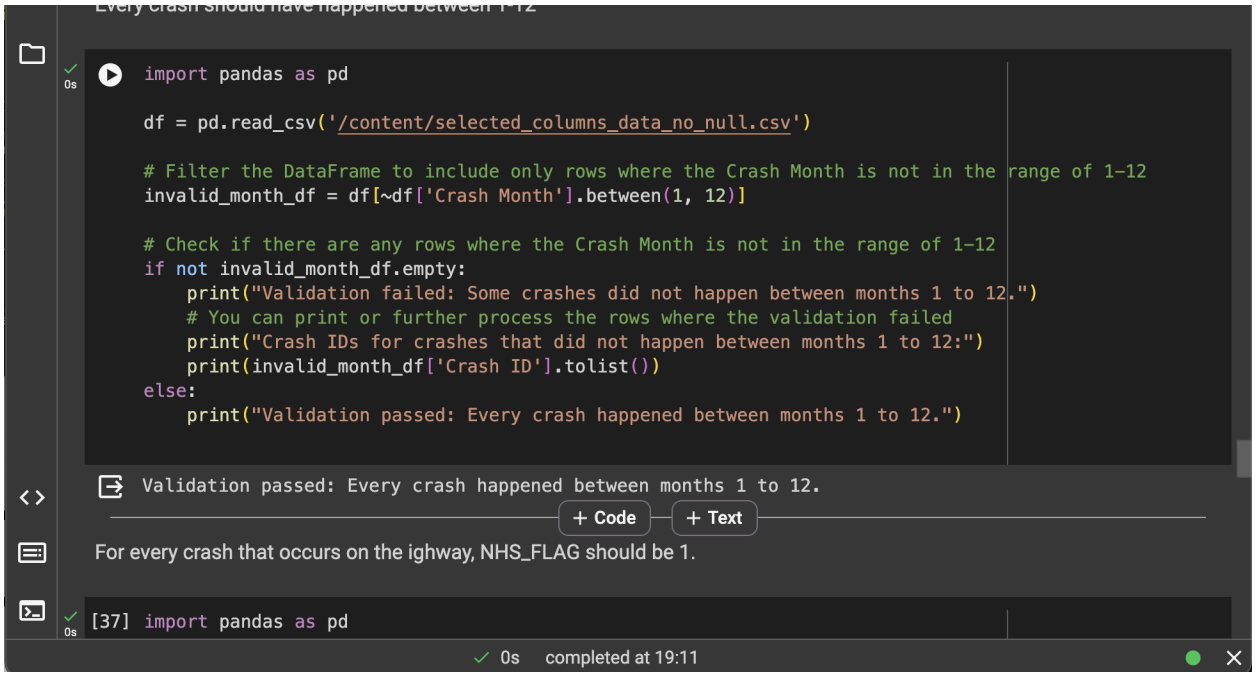


2. *limit* assertions. Example: “Every crash occurred during the year 2019”

a.Every crash occurred on Highway 26: validation passed



B. **`Every crash should have happened between month 1-12** : validation passed

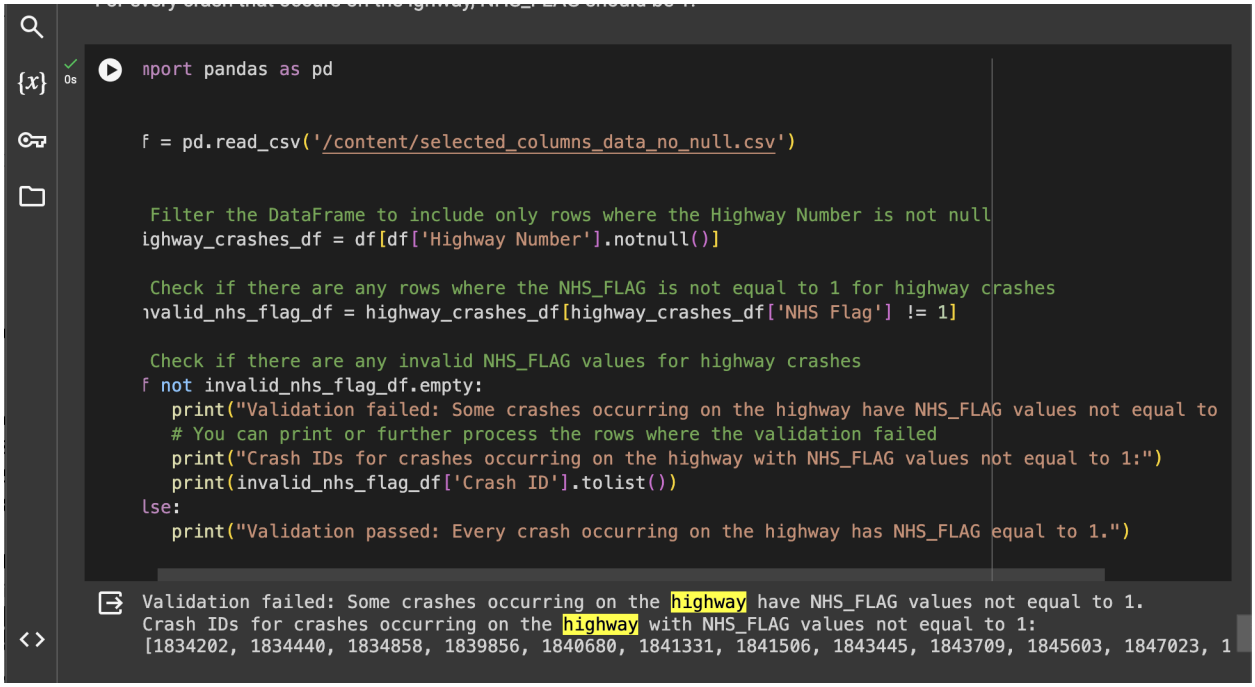


3. *intra-record* assertions. Example: “If a crash record has a latitude coordinate then it should also have a longitude coordinate”

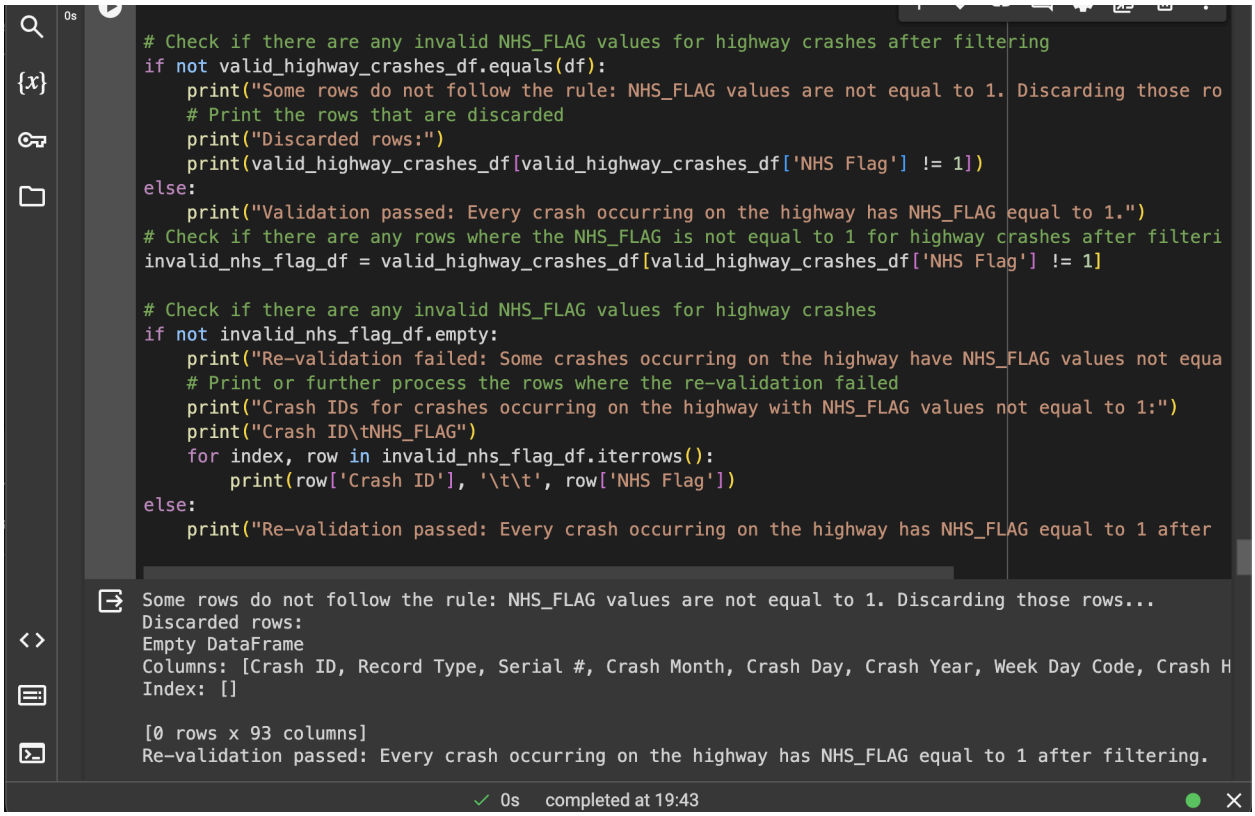
a. For every crash that occurs on the Highway, NHS_FLAG should be 1.

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]



Solution: as our data represents accident happening in NH 26 so will discard the rows not following this rule



```
# 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 ro
    # 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 filteri
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 equa
    # 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
```

```
Some rows do not follow the rule: NHS_FLAG values are not equal to 1. Discarding those rows...
Discarded rows:
Empty DataFrame
Columns: [Crash ID, Record Type, Serial #, Crash Month, Crash Day, Crash Year, Week Day Code, Crash H
Index: []

[0 rows x 93 columns]
Re-validation passed: Every crash occurring on the highway has NHS_FLAG equal to 1 after filtering.
```

0s completed at 19:43

b. 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).
```

0s completed at 20:45

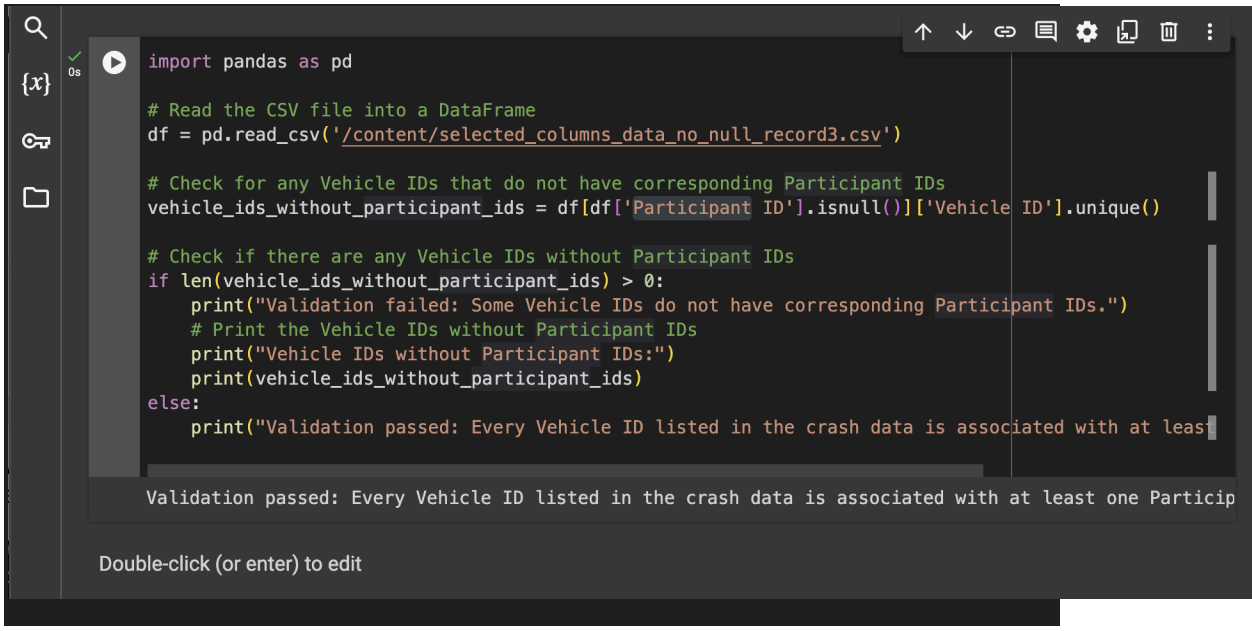
4. Create 2+ *inter-record check* assertions. Example: “Every vehicle listed in the crash data was part of a known crash”
    a. **If there is a crash ID there must be a record type.**

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

```python
# 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.
```

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



```
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 leas
```

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

Double-click (or enter) to edit

5. Create 2+ *summary* assertions. Example: “There were thousands of crashes but not millions”
    a. All crashes should occur in Oregon

```
Validation failed: Some entries with 'Speed Involved Flag' equal to 1 have
'State' column values other than 'OR'.
Crash IDs for the failed rows:
[1809119, 1809229, 1809637, 1810874, 1812266, 1815964, 1816804, 1826266,
1826321, 1826673, 1826794, 1826971, 1827119, 1827863, 1827872, 1827907,
1828231, 1828287, 1828314, 1828373, 1828552, 1828709, 1828977, 1828982,
1831161, 1831660, 1831698, 1832302, 1832348, 1833315, 1833381, 1833535,
1833867, 1833899, 1833971, 1833979, 1833986, 1834029, 1834096, 1834202,
1834407, 1834440, 1834574, 1834655, 1834846, 1834858, 1835061, 1835182,
1835197, 1835205, 1835211, 1835217, 1835225, 1835231, 1835315, 1835316,
1835322, 1835330, 1835332, 1835374, 1835382, 1835386, 1835459, 1835471,
1835511, 1835592, 1835606, 1835807, 1836788, 1837049, 1837085, 1837128,
1837131, 1837205, 1837283, 1837334, 1837497, 1837551, 1837808, 1837834,
1838047, 1838362, 1838375, 1838403, 1838434, 1838471, 1838549, 1838681,
1839606, 1839856, 1840093, 1840135, 1840155, 1840221, 1840265, 1840295,
1840424, 1840563, 1840583, 1840588, 1840610, 1840614, 1840664, 1840680,
1840832, 1840874, 1840905, 1841009, 1841013, 1841019, 1841052, 1841151,
1841157, 1841212, 1841331, 1841336, 1841343, 1841361, 1841369, 1841376,
1841436, 1841498, 1841506, 1841641, 1841685, 1841691, 1841693, 1841708,
```

1841725, 1841766, 1841806, 1841855, 1841914, 1842069, 1842180, 1842244,
1842388, 1842703, 1842741, 1842822, 1842862, 1843136, 1843196, 1843399,
1843400, 1843445, 1843562, 1843658, 1843709, 1843746, 1843752, 1843842,
1844151, 1844301, 1844339, 1844458, 1844484, 1844488, 1844762, 1844813,
1844857, 1844942, 1844976, 1845024, 1845085, 1845101, 1845113, 1845254,
1845255, 1845302, 1845326, 1845378, 1845474, 1845603, 1845619, 1845672,
1845727, 1845755, 1845835, 1845949, 1845964, 1845971, 1845975, 1845982,
1846015, 1846016, 1846021, 1846139, 1846145, 1846158, 1846237, 1846303,
1846318, 1846364, 1846373, 1846424, 1846442, 1846452, 1846543, 1846555,
1846575, 1846641, 1846699, 1846707, 1846797, 1846812, 1846840, 1846965,
1847023, 1847026, 1847080, 1847084, 1847184, 1847234, 1847302, 1847308,
1847313, 1847332, 1847352, 1847358, 1847366, 1847371, 1847378, 1847385,
1847483, 1847491, 1847500, 1847540, 1847575, 1847667, 1847722, 1847889,
1847898, 1847900, 1847935, 1847949, 1847958, 1848047, 1848054, 1848083,
1848156, 1848222, 1848270, 1848337, 1848347, 1848359, 1848489, 1848512,
1848553, 1848565, 1848616, 1848719, 1848792, 1848895, 1848913, 1849019,
1849023, 1849220, 1849271, 1849286, 1849289, 1849323, 1849337, 1849386,
1849472, 1849480, 1849501, 1849503, 1849552, 1849597, 1849637, 1849683,
1849777, 1849815, 1849893, 1849940, 1849971, 1850002, 1850038, 1850223,
1850230, 1850352, 1850368, 1850651, 1850653, 1850737, 1850744, 1850772,
1850791, 1850810, 1850874, 1850879, 1850898, 1850911, 1850939, 1850968,
1850983, 1851014, 1851035, 1851041, 1851062, 1851072, 1851075, 1851097,
1851131, 1851142, 1851178, 1851189, 1851193, 1851272, 1851274, 1851313,
1851365, 1851526, 1851540, 1851552, 1851557, 1851558, 1851564, 1851581,
1851609, 1851688, 1851693, 1851700, 1851706, 1851721, 1851723, 1851759,
1851776, 1851837, 1851849, 1851940, 1851959, 1851966, 1852005, 1852048,
1852070, 1852116, 1852143, 1852165, 1852201, 1852212, 1852223, 1852287,
1852367, 1852369, 1852380, 1852381, 1852384, 1852385, 1852404, 1852464,
1852483, 1852562, 1852623, 1852657, 1852691, 1852735, 1852769, 1852773,
1852807, 1852828, 1852853, 1852870, 1852889, 1852933, 1852934, 1852937,
1852965, 1852981, 1853011, 1853019, 1853209, 1853242, 1853265, 1853304,
1853339, 1853353, 1853448, 1853466, 1853499, 1853505, 1853514, 1853517,
1853519, 1853543, 1853582, 1853585, 1853606, 1853699, 1853728, 1853731,
1853788, 1853809, 1853845, 1853856, 1853862, 1853863, 1853879, 1853892,
1853898, 1853962, 1854007, 1854013, 1854063, 1854066, 1854071, 1854092,
1854099, 1854169, 1854273, 1854286, 1854309, 1854335, 1854338, 1854339,
1854347, 1854392, 1854398, 1854483, 1854569, 1854731, 1854794, 1854861,
1854881, 1854887, 1854987, 1854997, 1855110, 1855361, 1855400, 1855571,
1855657, 1855808, 1855937, 1855957, 1856211, 1856237, 1856784, 1857340,
1857364, 1857401, 1857514, 1857668, 1858032, 1858057, 1858160, 1858180,
1858328, 1858585, 1858607, 1858723, 1858727, 1858806, 1858869, 1858965,
1859192, 1859233, 1859239, 1859379, 1859422, 1859523, 1859565, 1859613,
1859616, 1859785, 1860007, 1860036, 1860043, 1860053, 1860257, 1860289,

1860371, 1860417, 1860427, 1860453, 1860771]

```
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 'State' column values not equal to 'OR'
invalid_state_df = df[~(df['Speed Involved Flag'] == 'OR')]

# Check if there are any rows with 'State' column values not equal to 'OR'
if not invalid_state_df.empty:
    print("Validation failed: Some entries with 'Speed Involved Flag' equal to 1 have 'State' column
    # Print Crash IDs for the failed rows
    print("Crash IDs for the failed rows:")
    print(invalid_state_df['Crash ID'].tolist())
else:
    print("Validation passed: All entries with 'Speed Involved Flag' equal to 1 have 'State' column
```

```
Validation failed: Some entries with 'Speed Involved Flag' equal to 1 have 'State' column values othe
Crash IDs for the failed rows:
[1809119, 1809229, 1809637, 1810874, 1812266, 1815964, 1816804, 1826266, 1826321, 1826673, 1826794, 1
```

Verify the average number of vehicles involved in each crash

Solution: 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.

```
# 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 Involve
    # 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
```

```
ion passed: All entries with 'Speed Involved Flag' equal to 1 have 'Speed Involved Flag' values equal
```

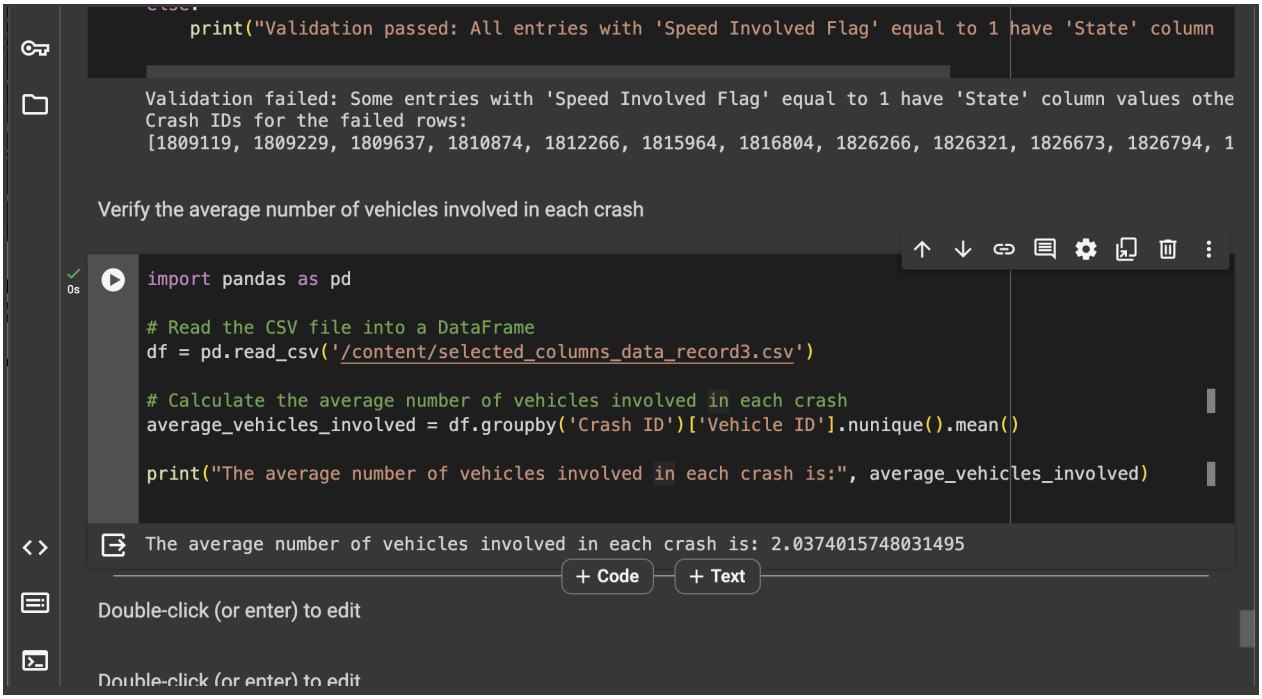
all states must be oregon

```
[61] import pandas as pd
```

0s completed at 19:53

b.Verify the average number of vehicles involved in each crash

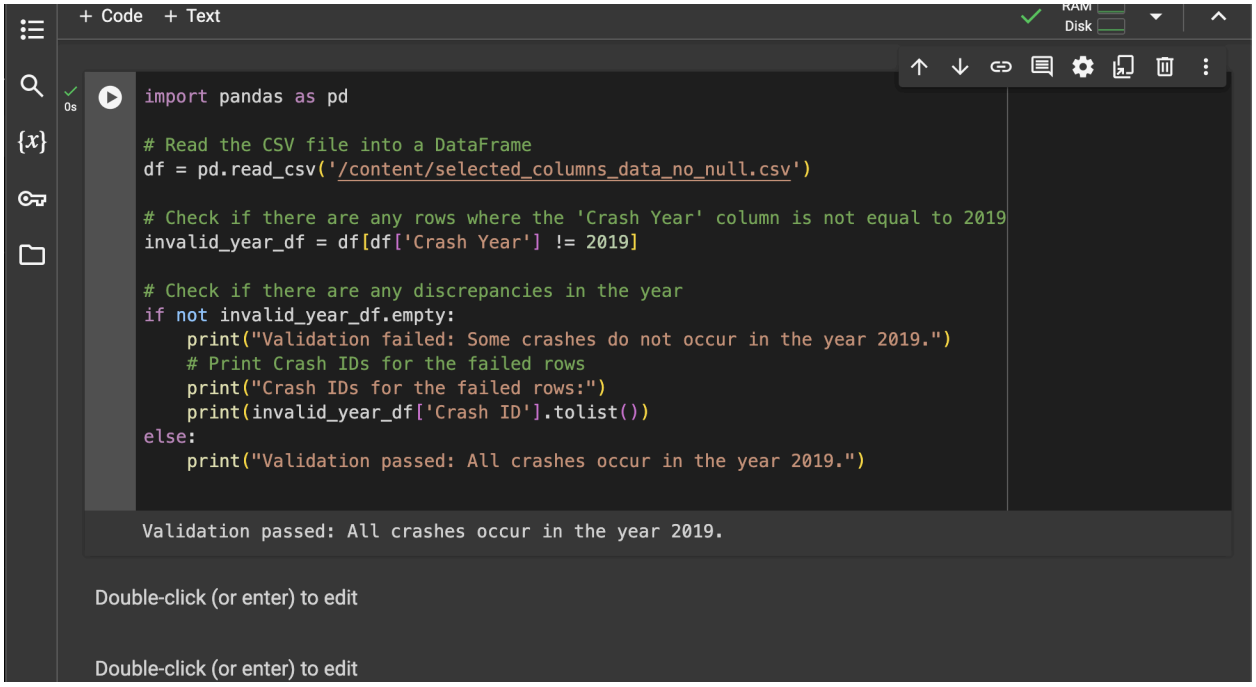
```
The average number of vehicles involved in each crash is:
2.0374015748031495
```



b.All crashes should occur in 2019 : Validation failed

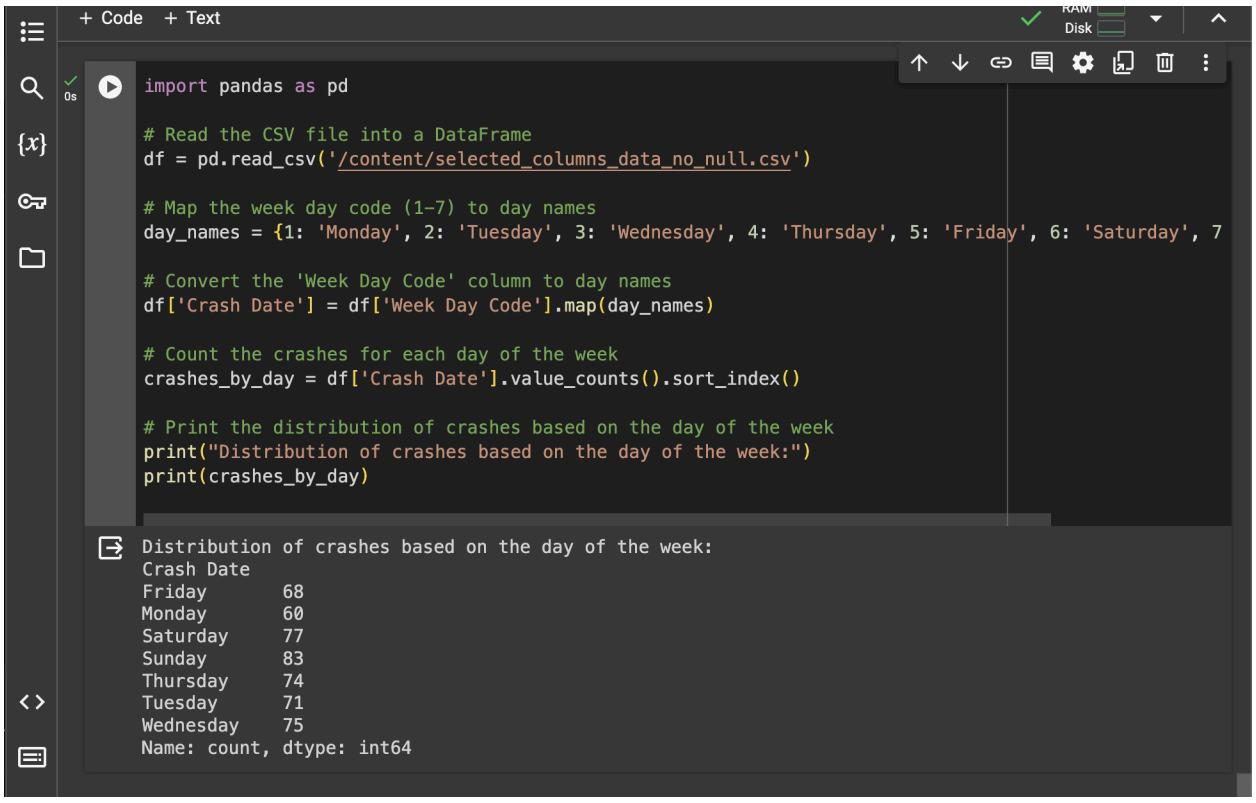
```
Checking Summary Assertion
Validation failed. The following records occur outside of 2019:
      Crash ID  Record Type  Vehicle ID  ...  Drug Use Reported  Participant Marijuana Use Reported  Participant Striker Flag
0      1809119            1         NaN  ...                NaN                                 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

[508 rows x 157 columns]
PS D:\PSU\All _Courses\Data Engineering\Inclass-Activity\Activity_Week4>
```

Divided the table based on record, this makes this validation pass

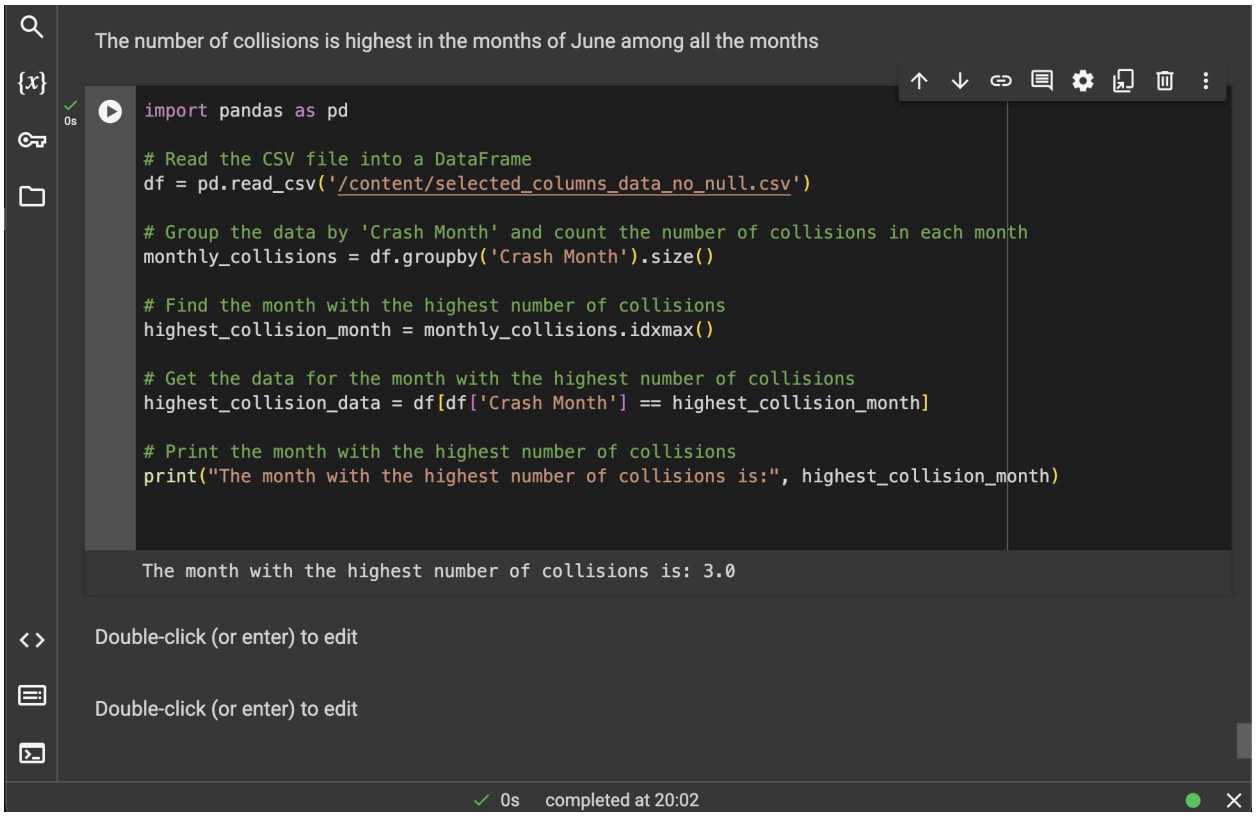


6.Create 2+ *statistical distribution assertions*. Example: “crashes are evenly/uniformly distributed throughout the months of the year.”

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



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



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.

Ans: Validated everything under B

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 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: 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: Transformed 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?

Ans: I learned about all the columns and their meaning, the form data is represented and its significance as well, chopping the data into 3 different files helped me to analyse each data set in a better way

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