DataEng: Data Validation Activity

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 for this week.

High quality data is crucial for any data project. This week you'll gain some experience and knowledge of analyzing data sets for quality.

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. First (part A) you develop assertions about your data as a way to make your assumptions explicit. Second (part B) you write code to evaluate the assertions and test the assumptions. This helps you to refine your existing assertions (part C) before starting the whole process over again by creating new assertions (part A again).

Submit: In-class Activity Submission Form

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 for this assignment, two or more assertions in each category are enough.

1. Create 2+ *existence* assertions. Example, "Every record has a date field".

   Every record has a date field. True
   100% of the date has Crash Id. True
   Each Crash ID can correspond to multiple Vehicle IDs and Participant IDs. True

2. Create 2+ *limit* assertions. The values of most numeric fields should fall within a valid range. Example: "the date field should be between 1/1/2019 and 12/31/2019 inclusive"

   The date field in Week Day Code should be between 1 and 7. True

```
count    508.000000
mean       4.145669
std        1.991709
min        1.000000
25%        2.000000
50%        4.000000
75%        6.000000
max        7.000000
Name: Week Day Code, dtype: float64
```

The date field in Crash hour is between 0 and 99. True

```
count    508.000000
mean      14.153543
std        9.131304
min        0.000000
25%       10.000000
50%       14.000000
75%       17.000000
max       99.000000
Name: Crash Hour, dtype: float64
```

The date field in Longitude Minutes should be between 1 and 59. True

```
count    508.000000
mean      31.474409
std        9.537290
min        1.000000
25%       29.000000
50%       33.000000
75%       38.000000
max       59.000000
Name: Longitude Minutes, dtype: float64
```

The data field in Sex should be 1,2,3,9 False, some of them is 4

```
count    1216.000000
mean        6.208059
std         2.675043
min         1.000000
25%         4.000000
50%         4.000000
75%         9.000000
max         9.000000
Name: Sex, dtype: float64
```

The data filed in Age should be 00,01,02-98,99 False, something wrong in this field.

```
count    1196.000000
mean        4.667224
std         3.414411
min         0.000000
25%         2.000000
50%         2.000000
75%         9.000000
max         9.000000
Name: Age, dtype: float64
```

The data filed in Weather Condition should be 0-9. False, just 0-5

```
count    508.000000
mean       1.562992
std        1.017581
min        0.000000
25%        1.000000
50%        1.000000
75%        2.000000
max        5.000000
Name: Weather Condition, dtype: float64
```

The Road Surface Condition should be 0-4. False, some is 99

```
count    508.000000
mean      36.751969
std       45.981060
min        0.000000
25%        1.000000
50%        4.000000
75%       99.000000
max       99.000000
Name: Road Surface Condition, dtype: float64
```

The Light Condition should be 0-5. False, just 1.

```
mean       1.0
std        0.0
min        1.0
25%        1.0
50%        1.0
75%        1.0
max        1.0
Name: Light Condition, dtype: float64
```

3. Create 2+ *intra-record check* assertions.

Total Fatality Count should bigger than total motorcyclists (operators and passengers) who were killed. False ,don't have a key called" total motorcyclists (operators and passengers) who were killed".

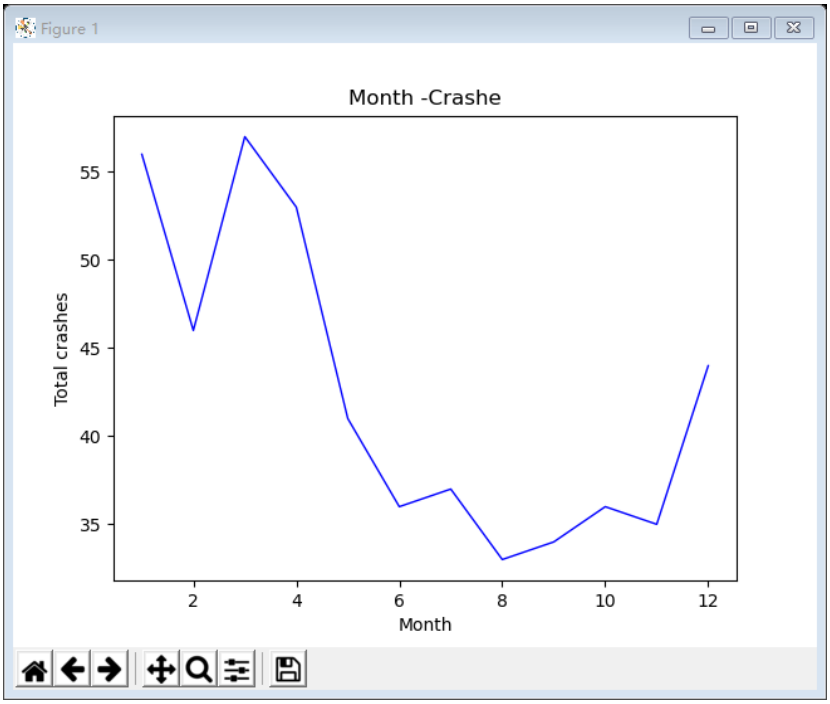
Total Unknown Participant Injury Count should less than Total Unknown ParticipantFatality Count. False, don't have such two keys

Total Pedestrians Involved in the Crash=Total Fatally Injured Pedestrians+Total Seriously Injured Pedestrians+Total Non-Fatally Injured Pedestrians False,don't have such two keys.

Total Count of Persons Involved= Total Persons Using Safety Equipment+Total Persons Not Using Safety Equipment+ Total Persons Safety Equipment Use Unknown False

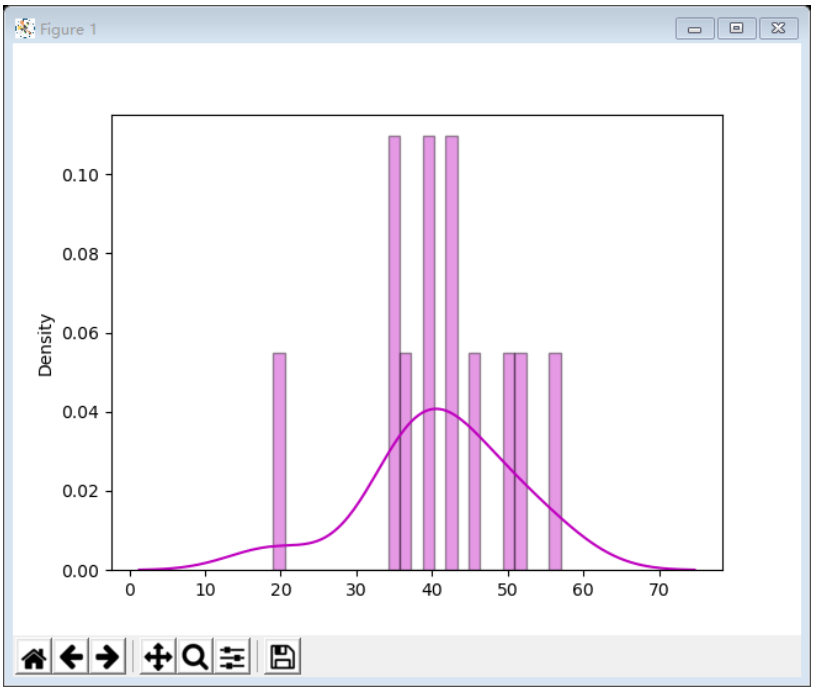
4. Create 2+ *inter-record check* assertions.

The number of crashes per month does not change by more than 10%.



False,(56-46)/56=17%>10%
Change it to:
The number of crashes per month does not change by more than 20%.

The total vehicle count of all crashes does not change more than 20% every month.

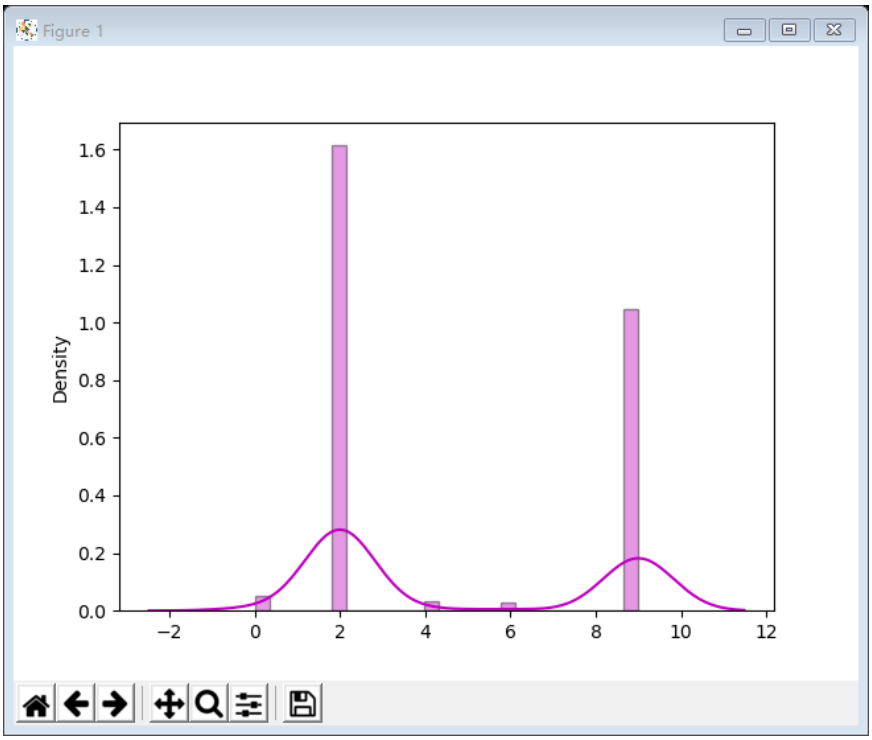


False, (55-36)/55=36%>20%

Change it to:

The total vehicle count of all crashes does not change more than 30% every month.

The age of participants does not change by more than 30%.



False,

Only have 5 kinds of ages, this part of data is wrong.

5. Create 2+ *summary* assertions. Example: “every crash has a unique ID”

   Crash Id should be unique,but it is not unique. True, because this table can be devided into 3 tables.
   Half of the data filed in each row is empty.
   Participant ID and Vehicle ID of a crash is not unique.

6. Create 2+ referential integrity insertions. Example “every crash participant has a Crash ID of a known crash”

   Every crash has a weather Condition Code of a known weather.
   True

Weather Condition EDW

Table: CRASH **Column:** WEATHR_COND_CD **Data Type:** char **Length:** 1

Code	Description
0	Unknown
1	Clear
2	Cloudy
3	Rain
4	Sleet / Freezing Rain / Hail
5	Fog
6	Snow
7	Dust
8	Smoke
9	Ash

   Every crash has a Crash Alcohol Involved Indicator shows whether the accident is related to alcohol.
   False, this tabel does not have this key.

   Every crash has a Road Surface Condition Code of a known road surface condition.

True

Road Surface Condition

Table: CRASH **Column:** RD_SURF_COND_CD **Data Type:** char **Length:** 1

Code	Description
0	Unknown
1	Dry
2	Wet
3	Snow
4	Ice

Description:

Road Surface Condition represents the condition of the travel lanes at the time of the crash.

Each Crash has a Light Condition of the amount of ambient light available at the time of the crash.

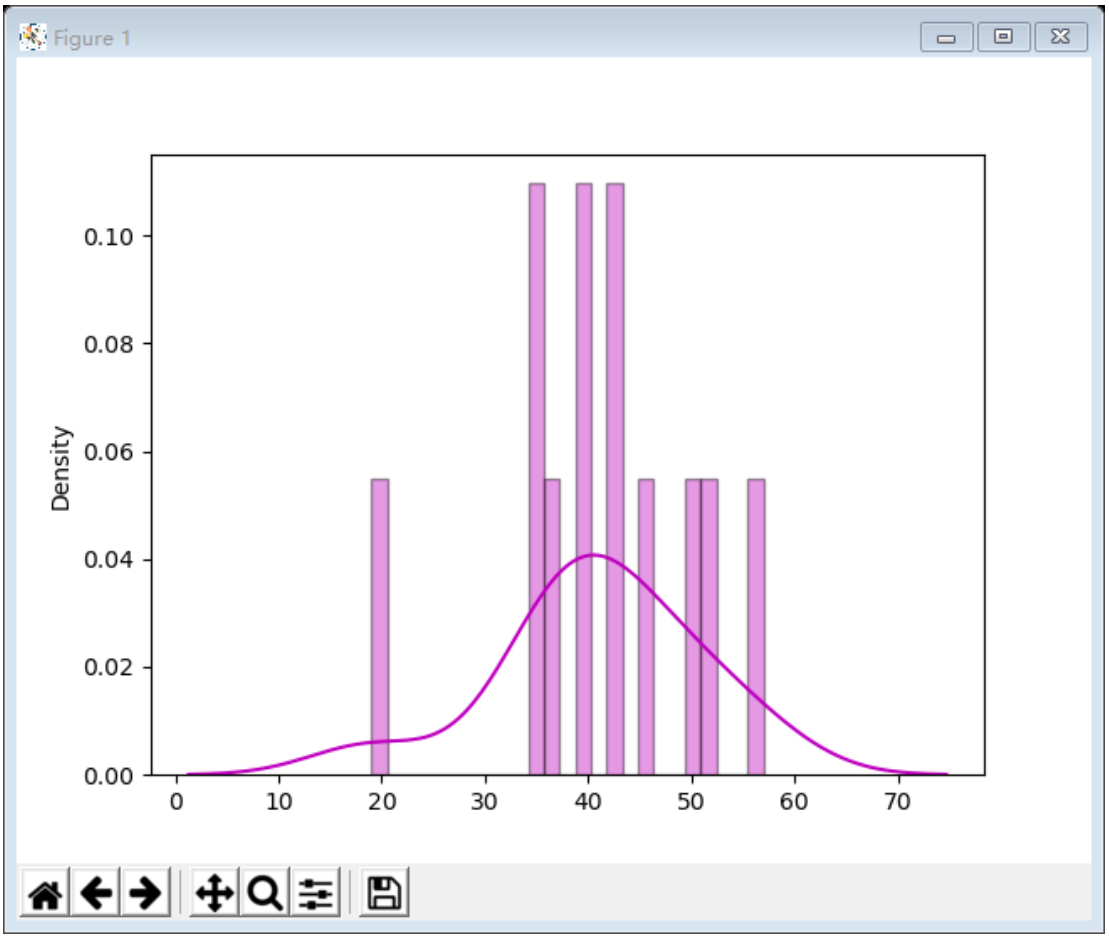
Light Condition

Table: CRASH **Column:** LGT_COND_CD **Data Type:** char **Length:** 1

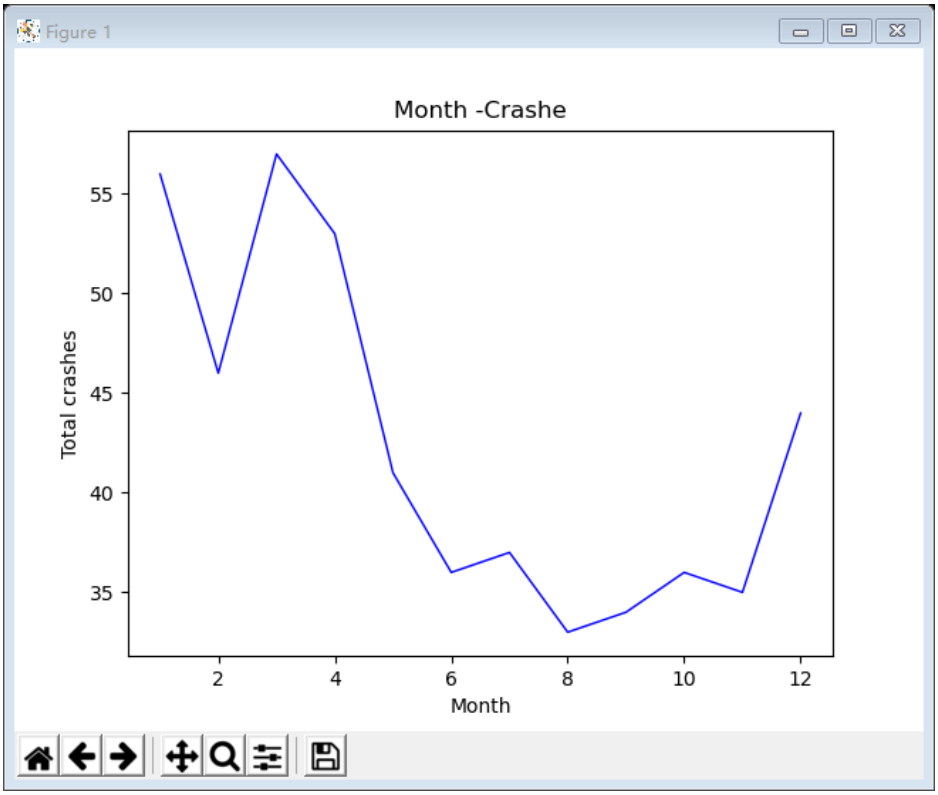
Code	Description
0	Unknown
1	Daylight
2	Darkness – with street lights
3	Darkness – no street lights
4	Dawn (Twilight)
5	Dusk (Twilight)

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

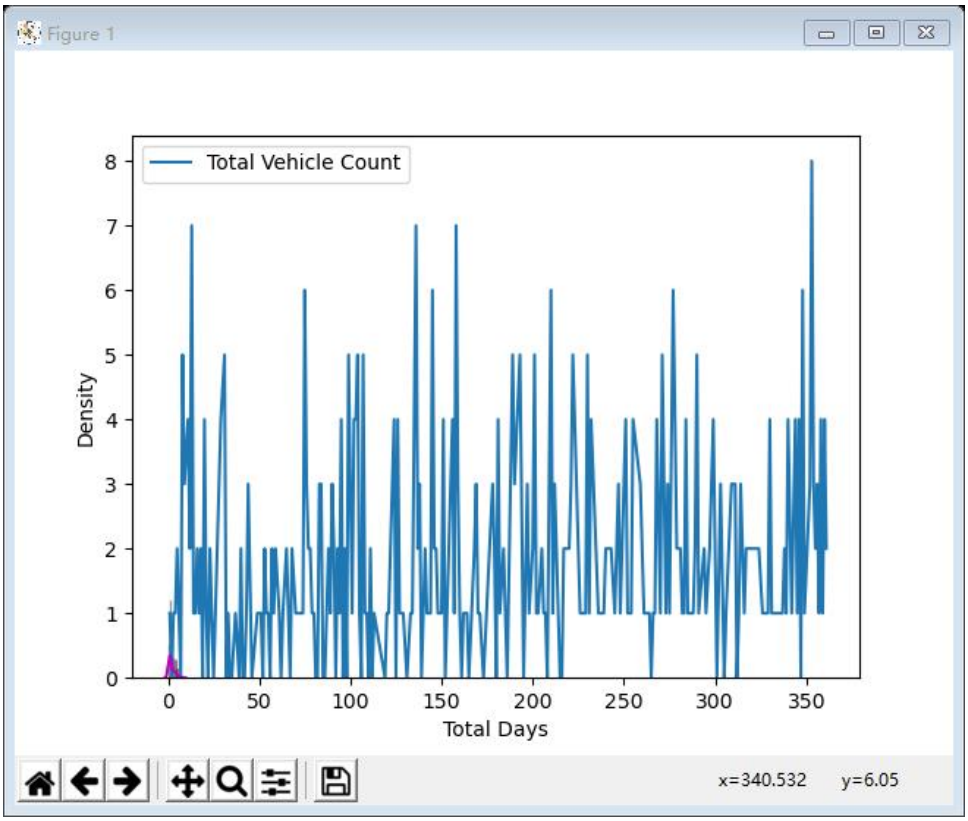
   The total vehicle count of all crashes in each month is evenly distributed throughout the year.



The sum of all crashes each month is evenly distributed throughout the year.
List:



The total vehicle count of all crashes in each day is evenly distributed throughout the year.



B. Validate the Assertions

1. Now study the data in an editor or browser. If you are anything like me you will be surprised with what you find. The Oregon DOT made a mess with their data!
2. Write python code to read in the test data and parse it into python data structures. You can 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. Again, pandas makes it easy to create and execute assertion validation code.
4. If you are like me you'll find that some of your assertions don't make sense once you actually understand the structure of the data. So go back and change your assertions if needed to make them sensible.
5. Run your code and note any assertion violations. List the violations here.

C. Evaluate the Violations

For any assertion violations found in part B, describe how you might resolve the violation. Options might include "revise assumptions/assertions", "discard the violating row(s)", "ignore", "add missing values", "interpolate", "use defaults", etc.

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

If you chose to "revise assumptions/assertions" for any of the violations, then briefly explain how you would revise your assertions based on what you learned.

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 ABCD iteration?

Next, iterate through the process again by going back to Step A. Add more assertions in each of the categories before moving to steps B and C again. Go through the full loop twice before moving to step E.

E. Resolve the Violations

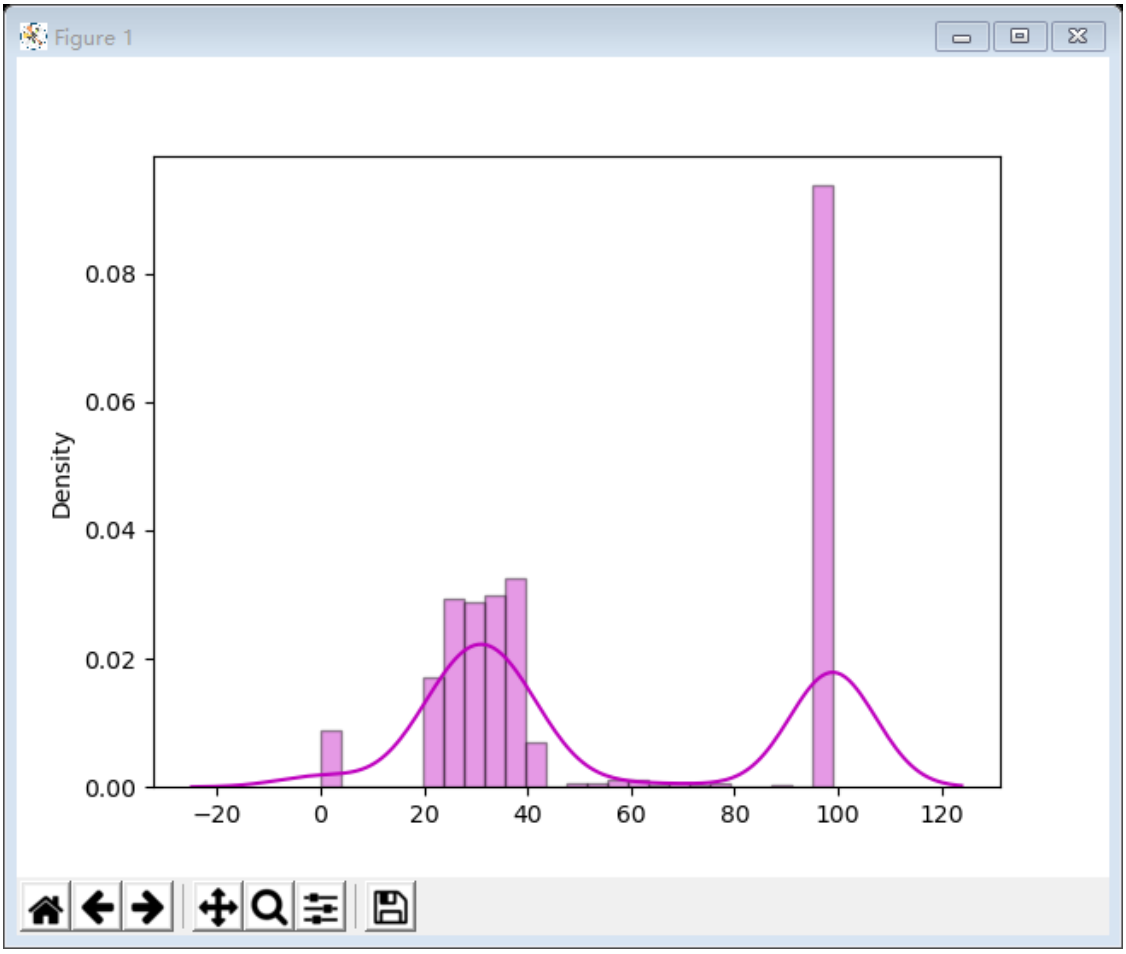
For each assertion violation found during the two loops of the process, write python code to resolve the assertions. This might include dropping rows, dropping columns,

adding default values, modifying values or other operations depending on the nature of the violation.

After I found something wrong with the ages, I try to modify values:

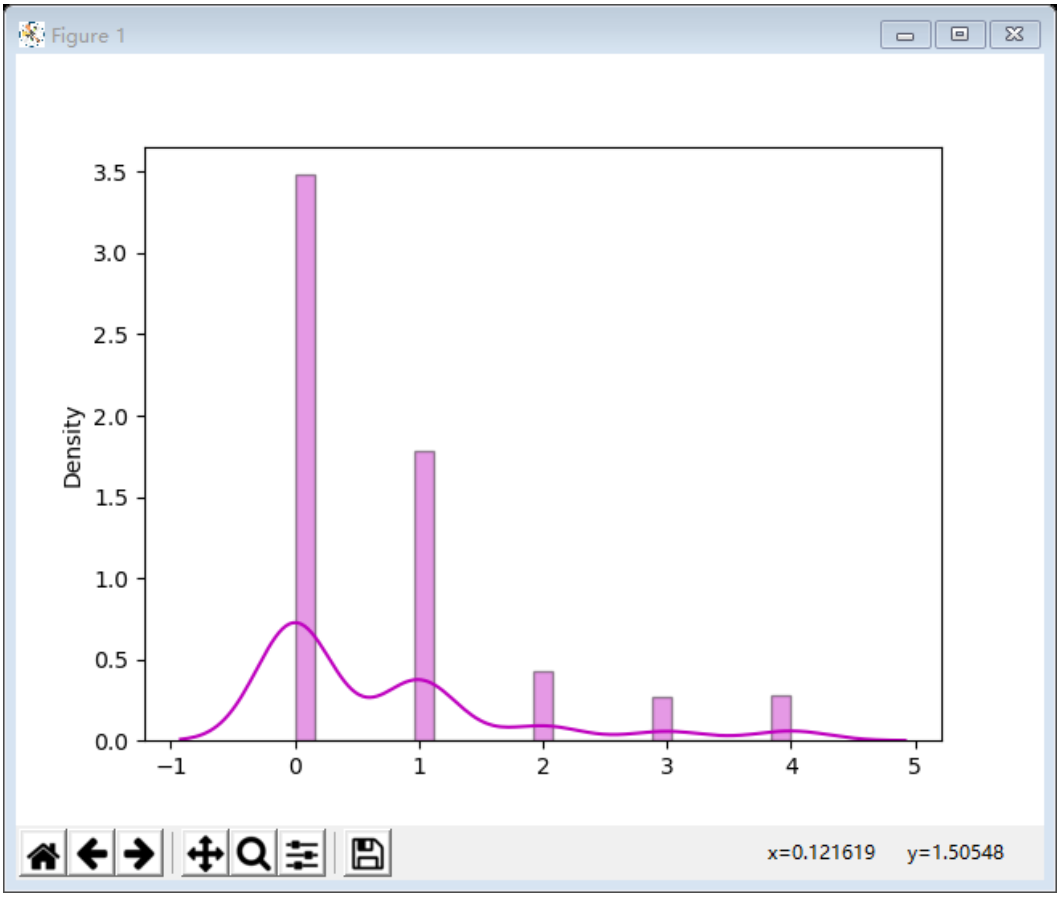
```
ages = ParticipantsDF["Age"].tolist()
for i in range(len(ages)):
    if (ages[i] == 0):
        ages[i] = "00"
    elif (ages[i] == 2):
        ages[i] = str(20 + random.randint(1, 20))
    elif (ages[i] == 4):
        ages[i] = str(40 + random.randint(1, 20))
    elif (ages[i] == 6):
        ages[i] = str(60 + random.randint(1, 38))
    elif (ages[i] == 9):
        ages[i] = "99"
    else:
        ages[i] = "00"
print(ages)
ParticipantsDF["Age"] = ages
```

After that, still have some ages is too high,but better than before.



I try to modify the Road Surface Condition:

```
conditions=CrashesDF["Road Surface Condition"].tolist()
for i in range(len(conditions)):
    if conditions[i] not in range(0,4):
        if (conditions[i]==99):
            conditions[i]=0
        else:
            conditions[i]=random.randint(0,4)
print(conditions)
CrashesDF["Road Surface Condition"]=conditions
```

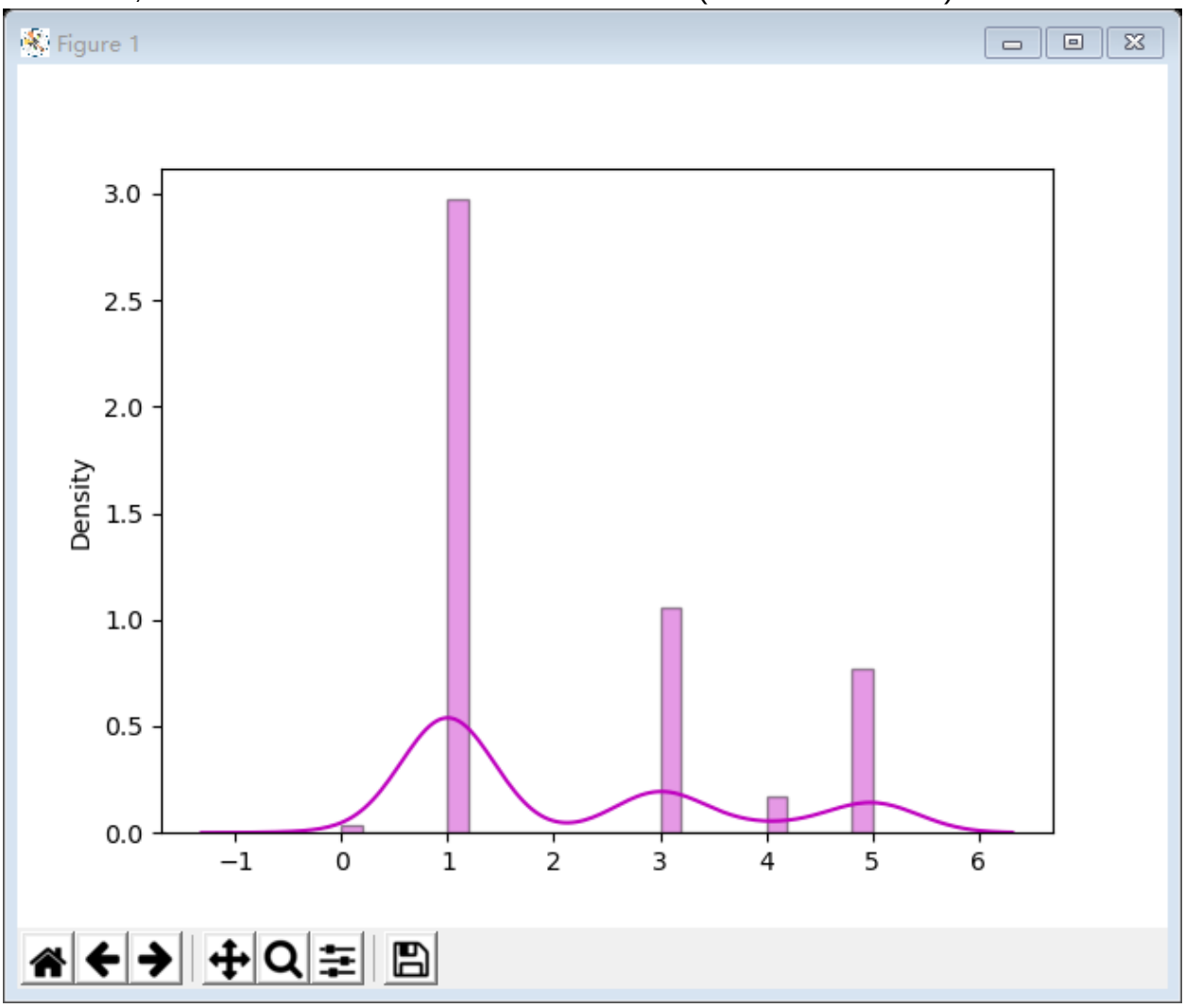


I try to modify the light condition by crash hour.

```
crash_hour=CrashesDF["Crash Hour"].tolist()
light_conditions=[]
for i in range(len(crash_hour)):
    if(crash_hour[i]==99):
        light_conditions.append(0)
    elif crash_hour[i] in range(5,7):
        light_conditions.append(4)
    elif crash_hour[i] in range(17,19):
        light_conditions.append(5)
    elif crash_hour[i] in range(7,17):
        light_conditions.append(1)
    else:
        light_conditions.append(3)
```

```
print(light_conditions)
CrashesDF["Light Condition"]=light_conditions
```

After that, I found that values are better than before(all of values are 1):



Note that I realize that this data set is somewhat awkward and that it might be best to “resolve the violations” by restructuring the data into proper tables. However, for this week, I ask that you keep the data in its current overall structure. Later (next week) we will have a chance to separate vehicle data and participant data properly.

E. Retest

After modifying the dataset/stream to resolve the assertion violations you should have produced a new set of data. Run this data through your validation code (Step B) to make sure that it validates cleanly.

Submit: In-class Activity Submission Form