Project: Capstone Project 1 – (Data Wrangling Exercise)

Project name: -	Explanatory Analysis of Traffic pullover pattern for Florida v/s Vermont
Student Name: -	Jitendra Agarwal
Course: -	Springboard cohort Jan2 2018
Data Source: -	https://openpolicing.stanford.edu/data/
Data provider: -	Openpolicing project by Stanford

About the DATA

The Raw data for this project contains the traffic stop data collected for 30+ states for open police project by Stanford research team. Standardized stop data are available to download (by state) from the link above provided by Stanford.
The csv includes a subset of common fields for each state, and indicates whether data are available for at least 70% of records in that state. Some states have more fields.
The original, unprocessed data we collected contain even more information.
The Stanford Open Policing Project data are made available under the Open Data Commons Attribution License.

Downloaded excel sheet of raw data for VT: -
https://github.com/jiagarwa/capstone-project1-Jitendra
file name: - 'VT-clean.csv.gz'

Accessing and filtering the Data

1. Read data from csv file VT-clean.csv and Show the all columns of data and their data type.

```
In [24]: import pandas as pd
         data = pd.read_csv('data/VT-clean.csv')
         df = pd.DataFrame(data)

         df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 283285 entries, 0 to 283284
Data columns (total 23 columns):
id                      283285 non-null object
state                   283285 non-null object
stop_date               283285 non-null object
stop_time               283285 non-null object
location_raw            282591 non-null object
county_name             282580 non-null object
county_fips             282580 non-null float64
fine_grained_location   282938 non-null object
police_department       283285 non-null object
driver_gender           281573 non-null object
driver_age_raw          282114 non-null float64
driver_age              281999 non-null float64
driver_race_raw         279301 non-null object
driver_race             278468 non-null object
violation_raw           281107 non-null object
violation               281107 non-null object
search_conducted        283285 non-null bool
search_type_raw         281045 non-null object
search_type             3419 non-null object
contraband_found        283251 non-null object
stop_outcome            280960 non-null object
is_arrested             283285 non-null bool
officer_id              283273 non-null float64
dtypes: bool(2), float64(4), object(17)
memory usage: 45.9+ MB
```

2. Filter data for Year 2015: -
   Convert data into a data frame and filter based on year from stop_date column and save in a separate file Filter data by year 2015. Call it VT_2015.csv

```
In [25]: #Filter 2015 Traffic data based on year in the stop_date column
traffic_2015 = df.loc[pd.to_datetime(df['stop_date']).dt.year == 2015]

traffic_2015.to_csv('data/VT_2015.csv')
```

What are the common problems found in the data?

- Check Missing Data
- Remove Duplicate rows
- Detect Outliers using Data visualization
- Untidy data
- Drop rows with missing key data

- Missing data and checking data types for all data

  Using info() method on data frame we checked the summary to see whether any data is missing. As you see below the number of null value for “search_type” is significantly high and rest all columns are populated more than 95% row.

```
In [28]: traffic_2015.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 45662 entries, 237623 to 283284
Data columns (total 23 columns):
id                      45662 non-null object
state                   45662 non-null object
stop_date               45662 non-null object
stop_time               45662 non-null object
location_raw            45634 non-null object
county_name             45631 non-null object
county_fips             45631 non-null float64
fine_grained_location   45623 non-null object
police_department       45662 non-null object
driver_gender           45440 non-null object
driver_age_raw          45143 non-null float64
driver_age              45119 non-null float64
driver_race_raw         44984 non-null object
driver_race             44745 non-null object
violation_raw           45479 non-null object
violation               45479 non-null object
search_conducted        45662 non-null bool
search_type_raw         45470 non-null object
search_type             591 non-null object
contraband_found        45660 non-null object
stop_outcome            45447 non-null object
is_arrested             45662 non-null bool
officer_id              45654 non-null float64
dtypes: bool(2), float64(4), object(17)
memory usage: 7.8+ MB
```

- <u>Remove Duplicate rows</u>

  As we can see there is a unique column “id” already exists in data and there are no duplicate rows to cleanup further. Each record of the csv file represents a unique traffic stop.

```
In [30]: traffic_2015[traffic_2015.duplicated(['id'])]
Out[30]:
```

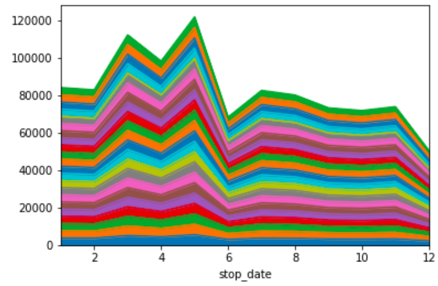
id	state	stop_date	stop_time	location_raw	county_name	county_fips	fine_grained_location	police_department	driver_gender	...	driver_race	violation_rav

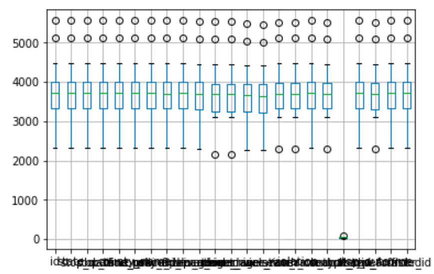
0 rows × 23 columns

- <u>Detect Outliers and data anomalies using Data visualization</u>

  Using area plot and box plot we checked the data population is consistent across the month (a given window) and the reported incident number of traffic stop in each month does looks fine. There are no outliers and there are no anomalies or no missing data as a big chunk.

```
In [49]: monthly_traffic = traffic_2015.groupby(pd.to_datetime(traffic_2015['stop_date']).dt.month).count()
         monthly_traffic.plot.area(legend = False )
         import matplotlib.pyplot as plt
         plt.show()
         monthly_traffic.boxplot()
         plt.show()
```





- <u>Tidy Data</u>

  - Columns represent separate variables
  - Rows represent individual observations

- <u>Drop rows with missing key data</u>

  Drop records where the traffic stop reason or the stop outcome is unknown. These records will not be useful and any meaningful analysis.
  As you can see below out of 45662 rows, 45423 rows were remained and 239 records were dropped.

```
In [50]: traffic_2015.dropna(subset = ['violation', 'stop_outcome'])
```

283280	VT-2015-45658	VT	2015-12-31	23:09	Hartford	Windsor County	50027.0	I 91 S MM69	ROYALTON VSP	M	...	White
283281	VT-2015-45659	VT	2015-12-31	23:40	Hartland	Windsor County	50027.0	I 91 S MM66	ROYALTON VSP	F	...	White
283282	VT-2015-45660	VT	2015-12-31	23:44	Brattleboro	Windham County	50025.0	MAIN ST & HIGH ST	BRATTLEBORO VSP	F	...	White
283283	VT-2015-45661	VT	2015-12-31	23:55	Weathersfield	Windsor County	50027.0	VT RT 131 & VT RT 106	ROCKINGHAM VSP	M	...	White
283284	VT-2015-45662	VT	2015-12-31	23:55	Brattleboro	Windham County	50025.0	768 PUTNEY RD; by Staples	BRATTLEBORO VSP	M	...	White

45423 rows × 23 columns

Aggregate data for further analysis

Most of the reports in exploratory analysis will be done based on logical grouping of data by date, time and other factors So aggregating data upfront is also useful.

- Aggregate by time of the day (slot of 4 hours)
- Aggregate by day of the week
- Aggregate by month
- Aggregate by age range (each slot of 10 years)

We will use pivot table features for aggregating data based on these criteria for analysis.

- Aggregate by month

```
In [65]: VT_agg_month = pd.pivot_table(VT_traf_2015_main, values='id', index=[pd.to_datetime(VT_traf_2015_main['stop_date']).dt.r
         print(VT_agg_month)

driver_gender     F     M
stop_date
1              1334  2465
2              1240  2509
3              1809  3265
4              1639  2793
5              2094  3404
6              1113  1977
7              1391  2345
8              1304  2318
9              1219  2096
10             1210  2030
11             1250  2105
12              794  1504
```

- Aggregate by age range (each slot of 10 years)

  We first aggregate by age and then this result data frame can be used in a histogram with a slot size of 5 or 10 years.

```
In [67]: import numpy as np
VT_agg_age = pd.pivot_table(VT_traf_2015_main, values='id', index=['driver_age'], columns=['driver_gender'],
                            aggfunc=np.count_nonzero)
#VT_agg_age = pd.pivot_table(VT_traf_2015_main, values='id', index=np.floor_divide(VT_traf_2015_main['driver_age'], 5),
print(VT_agg_age)
```

```
driver_gender      F      M
driver_age
15.0             5.0    6.0
16.0            56.0   83.0
17.0           201.0  339.0
18.0           326.0  608.0
19.0           453.0  890.0
20.0           496.0  894.0
21.0           504.0  940.0
22.0           567.0  854.0
23.0           543.0  891.0
24.0           556.0  916.0
25.0           509.0  854.0
26.0           448.0  830.0
27.0           465.0  758.0
28.0           472.0  739.0
29.0           420.0  715.0
30.0           421.0  659.0
31.0           420.0  615.0
32.0           354.0  607.0
33.0           325.0  574.0
34.0           305.0  581.0
35.0           313.0  554.0
36.0           305.0  507.0
37.0           278.0  456.0
38.0           288.0  487.0
39.0           284.0  454.0
40.0           264.0  500.0
41.0           284.0  448.0
42.0           262.0  463.0
43.0           230.0  463.0
44.0           271.0  491.0
...              ...    ...
...              ...    ...
66.0           111.0  208.0
67.0           100.0  212.0
68.0           111.0  235.0
69.0            72.0  182.0
70.0            67.0  137.0
71.0            60.0  123.0
72.0            62.0  137.0
73.0            47.0   97.0
74.0            31.0   80.0
75.0            29.0   65.0
76.0            24.0   64.0
77.0            33.0   61.0
78.0            26.0   54.0
79.0            18.0   30.0
80.0            20.0   35.0
81.0            16.0   32.0
82.0             7.0   30.0
83.0             7.0   28.0
84.0             8.0   17.0
85.0             9.0   17.0
86.0             6.0    8.0
87.0             7.0   13.0
88.0             3.0   12.0
89.0             4.0    5.0
90.0             2.0    6.0
91.0             NaN    2.0
92.0             NaN    3.0
94.0             NaN    1.0
95.0             NaN    3.0
96.0             1.0    2.0

[81 rows x 2 columns]
```

- Aggregate by time of the day (slot of 4 hours)

  We first aggregate by hour and then this result data frame can be used in a histogram with a slot size of 4 or 6 hours.

```
In [77]: VT_agg_time = pd.pivot_table(VT_traf_2015_main, values='id', index=pd.to_datetime(VT_traf_2015_main['stop_time'])
                             .dt.hour, columns=['driver_gender'], aggfunc=np.count_nonzero)
         print(VT_agg_time)
```

```
driver_gender     F     M
stop_time
0               504  1151
1               278   737
2               118   376
3                27   105
4                 9    46
5                20    81
6                97   195
7               475   659
8               856  1229
9               886  1485
10              798  1238
11              606   948
12              448   812
13              697  1195
14              990  1705
15             1171  1916
16             1124  1875
17             1771  2909
18             1861  3132
19             1089  1943
20              626  1173
21              631  1203
22              683  1327
23              632  1371
```

- Aggregate by day of the week

```
In [79]: VT_agg_day = pd.pivot_table(VT_traf_2015_main, values='id', index=pd.to_datetime(df['stop_date']).
                             dt.dayofweek, columns=['driver_gender'], aggfunc=np.count_nonzero)
         print(VT_agg_day)
```

```
driver_gender     F     M
stop_date
0              2423  3803
1              2238  3665
2              2379  4009
3              2308  4017
4              2823  5094
5              2271  4449
6              1955  3774
```

So far we have created aggregated dataframes for further analysis.

VT_agg_age
VT_agg_month
VT_agg_time
VT_agg_day