Study of Open Parking and Camera violations using R & Python

Contents

Table of Figures

Executive Summary:

Fast-growing vehicles and limited parking lots have caused Open Parking and Camera Violation problems. Parking disorderly not only brings inconvenience to traffic flow but also has a negative impact on the image of the city. Our research is to evaluate several factors that are associated with the violations that happened. From our analysis, we would come up with improvements for pedestrians, vehicular safety through education and a timely decision of Parking, and traffic violations. Further, it would help the government to resolve violation issues in a systematic manner.

Statement of Scope:

The dataset reflects the daily volume of Open parking and Camera violations that have been recorded by each camera installed and reviewed by city contractors in different states. This dataset contains all violations regardless of a citation was issued. It also provides the original ticket violation numbers that were given to the violators. The data reflects violations that occurred from January 1, 2000, until December 31, 2017. Below are few of the evaluating factors:

1. What time of the day has the maximum number of tickets were issued?
2. What types of tickets are mostly issued?
3. Which states get the most tickets?
4. Which county has the highest concentration of parking violations?
5. What type of violations are mostly happened?
6. How does the violation trend differ from season to season?
7. What type of violations are recorded past midnight?

Project Objectives:

The project deals with the below-mentioned objectives:

To understand and evaluate several factors that are associated with the violations that happened between the Period January 1, 2000, to December 31, 2017.

Target and Predictor Variables:

To understand the important factors that are determining the Violation and predict which variables lead to Violation.

Target variables: Violation

Predictor Variables: Fine Amount, Payment Amount.

Project Schedule:

TASKS	January				February				March					April				May
	W1	W2	W3	W4	W1	W2	W3	W4	W1	W2	W3	W4	W5	W1	W2	W3	W4	W1
1.Dataset confirmation			7															
2.Project Proposal				3														
3.Scope Statement					3													
4.Data Access						2												
5.Data Consolidation						1	1											
6.Data Cleaning							3	4	2									
7.Data Transformation									4									
8.Data Reduction										3	5							
9.Data Statistics										1	1							
10.Data Dictionary											1							
11.Deliverable 1 - Submission											5							
12.Modeling Technique												3	3					
13.Build Model													5	5				

14.Assess Model														3	3			
15.Combine Deliverable 1 & Deliverable 2															3			
16.Final Report															3	1	1	
17.Final Presentation																1	1	
18.Presentation Record																	1	
19.Final Deliverable																		2

Figure 1: Gantt Chart

Work Breakdown Structure (WBS):

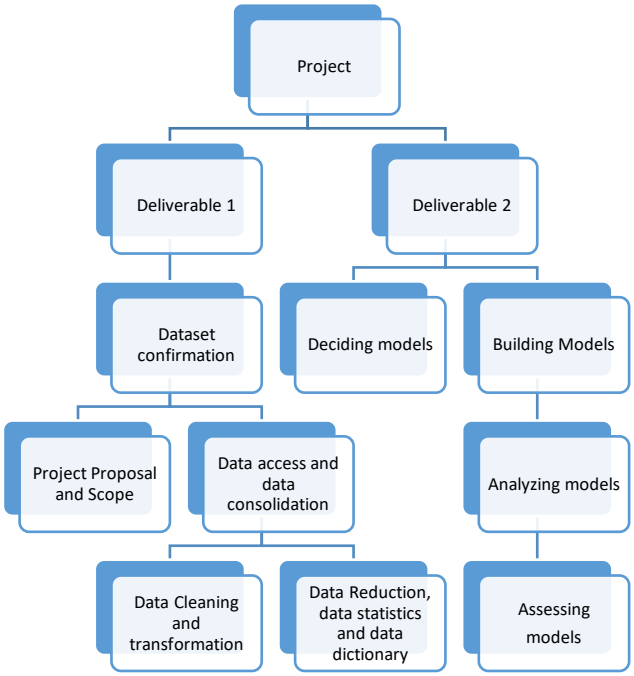

Figure 2: WBS

Resource Allocation:

Task Number	Task Name	Description of work	Number of resources and Name of the resource	Hours spent

1	Dataset confirmation	Confirming one dataset among the selected ones.	4(Team)	7
2	Project Proposal	Identifying the problem.	4(Team)	3
3	Scope statement	Finalizing the scope	4(Team)	3
4 & 5	Data Access and Consolidation	Getting the data from the source, combining data	1Mrunali)	4
6,7	Data cleaning & transformation	Clean the null values and transform any columns and rows	2(Chakri, & Bhavana)	13
8,9	Data reduction & statistics	Cleaning and removal of irrelevant data	1(Suveatha)	10
10 & 11	Data dictionary & Deliverable 1	Identifying and changing the data types, documenting the deliverable	1(Mrunali)	6
12	Modeling technique	Exploring the different modeling techniques	4(Team)	6
13	Build Model	Code and build the model decided	4(Team)	10
14	Assess Model	Evaluating the model and assessing them in detail	4(Team)	6
15	Combining Deliverable 1 & 2	Consolidating deliverable 1 with 2 and making modifications (if any) in deliverable 1	4(Team)	3
16	Final report	Documentation of the final report	4(Team)	6

Figure 3: Resource details

All the team members actively participated in all the biweekly meetings conducted. The meeting days were usually on Tuesdays and Thursdays of every week. In the Biweekly meetings, tasks were assigned to each of the team members and all the team members diligently completed their assigned tasks. This helped us to complete the project well in advance much before the project deadline.

Data Preparation:

Data Access:

We have obtained data from the city of New York open data. The data file we obtained is in the CSV format. The original dataset consists of 23.6 M records and 19 Variables, this data file contains Open Parking and Camera Violations issued by the City of New York for different states. The original file size was ~6 GB but the excel could handle only 136Mb with 1048575 records. So, we would be working on this limited number of records. Below is the link to the dataset and file: https://data.cityofnewyork.us/City-Government/Open-Parking-and-Camera- Violations/nc67

Data Consolidation:

The data we obtained from the NYC website was clean. We did not have to consolidate anything with respect to the Data consolidation section. We had the last column in our table that contained the image of the Summon issued to every person. Since the images of the summon are not going to be of any use in the analysis we decided to get rid of it and that was the only consolidation we did. In addition, we did not use python to get rid of it, Excel was good enough to get rid of the column.

Data Cleaning:

Columns with missing values:

The data initially had a lot of missing values in some of the columns. We filled those missing values with NaN first and dropped those columns. The below screenshot shows the count after the missing values were filled with NaN. The count was 1048575.

```
>>> violation_data1.isnull().count()
Plate               1048575
State               1048575
License Type        1048575
Summons Number      1048575
Issue Date          1048575
Violation Time      1048575
Violation           1048575
Fine Amount         1048575
Penalty Amount      1048575
Interest Amount     1048575
Reduction Amount    1048575
Payment Amount      1048575
Amount Due          1048575
County              1048575
Issuing Agency      1048575
```

Figure 4: Count

Next, we considered to delete the records that have Nan values and we found a total of 45,817 records with NaN data. The next screenshot shows the dropping of all the NaN values in data, the count was reduced to 1002758.

```
>>> violation_data1 = violation_data1.dropna()
>>> violation_data1.count()
Plate               1002758
State               1002758
License Type        1002758
Summons Number      1002758
Issue Date          1002758
Violation Time      1002758
Violation           1002758
Fine Amount         1002758
Penalty Amount      1002758
Interest Amount     1002758
Reduction Amount    1002758
Payment Amount      1002758
Amount Due          1002758
County              1002758
Issuing Agency      1002758
dtype: int64
```

Figure 5: Count after cleaning

We have five Amount columns with '$' symbol, we removed them using the following left strip method:

```
>>> #Strip '$' from the amount columns
... violation_data['Fine Amount'] = violation_data['Fine Amount'].str.lstrip('$')
...
>>> violation_data['Penalty Amount']=violation_data['Penalty Amount'].str.lstrip('$')
>>> violation_data['Interest Amount']=violation_data['Interest Amount'].str.lstrip('$')
>>> violation_data['Reduction Amount']=violation_data['Reduction Amount'].str.lstrip('$')
>>> violation_data['Payment Amount']=violation_data['Payment Amount'].str.lstrip('$')
>>> violation_data['Amount Due']=violation_data['Amount Due'].str.lstrip('$')
>>>
```

Figure 6: Dollar symbol stripping

Elimination of erroneous data:

We found our data had some erroneous entries, we figured out those entries using the unique method and eliminated them accordingly. Below are the faulty data we found:

- License Type had '999' as erroneous data
- The state had '99' as erroneous data
- Issue Date year was greater than 2018 which did not make sense as the data was updated last by end of 2017.
- Violation_Time data is in 12hour format, but we found certain records having data without AM and PM and hours greater than 12 hours.

Below screenshots explain how these above cases were handled.

```
>>> violation_data1['License Type'].unique()
array(['PAS', 'OMS', 'OMT', 'TRC', 'COM', 'IRP', 'APP', 'CMB', 'ORG',
       'ITP', 'CME', 'SRF', '999', 'DLR', 'MOT', 'MED', 'VAS', 'TOW',
       'OML', 'OMR', 'SRN', 'RGL', 'CMH', 'SCL', 'AYG', 'SPO', 'AGC',
       'TRA', 'HAM', 'OMF', 'MCL', 'CSP', 'PHS', 'CLG', 'CHC', 'PSD',
       'STG', 'HIS', 'TRL', 'AMB', 'SOS', 'SPC', 'RGC', 'ORC', 'OMV',
       'BOB', 'NYA', 'AGR', 'CBS', 'NYC', 'CCK', 'THC', 'FPW', 'VPL',
       'NLM', 'PPH', 'LTR', 'LUA', 'ARG', 'ORM', 'HAC', 'NYS', 'SEM',
       'USC', 'STA', 'MCD', 'ATD', 'LMA', 'USS', 'OMO'], dtype=object)
>>> violation_data2.drop(violation_data2[violation_data2['State'] == '99'].index, inplace=True)
>>>
>>> violation_data2['License Type'].unique()
array(['PAS', 'OMS', 'OMT', 'TRC', 'COM', 'IRP', 'APP', 'CMB', 'ORG',
       'ITP', 'CME', 'SRF', '999', 'DLR', 'MOT', 'MED', 'VAS', 'TOW',
       'OML', 'OMR', 'SRN', 'RGL', 'CMH', 'SCL', 'AYG', 'SPO', 'AGC',
       'TRA', 'HAM', 'OMF', 'MCL', 'CSP', 'PHS', 'CLG', 'CHC', 'PSD',
       'STG', 'HIS', 'TRL', 'AMB', 'SOS', 'SPC', 'RGC', 'ORC', 'OMV',
       'BOB', 'NYA', 'AGR', 'CBS', 'NYC', 'CCK', 'THC', 'FPW', 'VPL',
       'NLM', 'PPH', 'LTR', 'LUA', 'ARG', 'ORM', 'HAC', 'NYS', 'SEM',
       'USC', 'STA', 'MCD', 'ATD', 'LMA', 'USS', 'OMO'], dtype=object)
```

Figure 7: Unique Violation type

Both state and license columns have ‘99’ and ‘999’ respective entries which are erroneous and were removed using the following code and below is the screenshot showing the output:

```
>>> violation_data2=violation_data2[violation_data2.State != '99']
>>> violation_data2=violation_data2[violation_data2.License_Type != '999']
>>>
>>> violation_data2['License_Type'].unique()
array(['PAS', 'OMS', 'OMT', 'TRC', 'COM', 'IRP', 'APP', 'CMB', 'ORG',
       'ITP', 'CME', 'SRF', 'DLR', 'MOT', 'MED', 'VAS', 'TOW', 'OML',
       'OMR', 'SRN', 'RGL', 'CMH', 'SCL', 'AYG', 'SPO', 'AGC', 'TRA',
       'HAM', 'OMF', 'MCL', 'CSP', 'PHS', 'CLG', 'CHC', 'PSD', 'STG',
       'HIS', 'TRL', 'AMB', 'SOS', 'SPC', 'RGC', 'ORC', 'OMV', 'BOB',
       'NYA', 'AGR', 'CBS', 'NYC', 'CCK', 'THC', 'FPW', 'VPL', 'NLM',
       'PPH', 'LTR', 'LUA', 'ARG', 'ORM', 'HAC', 'NYS', 'SEM', 'USC',
       'STA', 'MCD', 'ATD', 'LMA', 'USS', 'OMO'], dtype=object)
>>> violation_data2['State'].unique()
array(['NY', 'PA', 'NJ', 'IL', 'IN', 'TX', 'SC', 'MN', 'MD', 'CT', 'NC',
       'GA', 'DE', 'OK', 'FL', 'MA', 'MO', 'AZ', 'IA', 'MI', 'RI', 'MS',
       'VA', 'ME', 'ON', 'VT', 'CA', 'CO', 'WV', 'OH', 'WA', 'LA', 'AL',
       'NV', 'NH', 'DC', 'OR', 'TN', 'QB', 'WI', 'KY', 'NE', 'DP', 'ID',
       'HI', 'KS', 'WY', 'ND', 'AR', 'MT', 'NM', 'SD', 'AB', 'AK', 'UT',
       'BC', 'GV', 'NS', 'PR', 'MB', 'PE', 'NB', 'SK', 'MX', 'FO'], dtype=object)
```

Figure 8: Violation Type

The Issue_Date column had inaccurate entries, we eliminated them and even restricted the date entries between January 1, 2000, to December 31, 2017, using the code shown below. There was a total of 5167 records with such entries.

```
>>> ts=pd.to_datetime('12/31/2017')
>>> violation_data2.loc[violation_data2.Issue_Date>ts,:].count()
Plate               5167
State               5167
License_Type        5167
Summons_Number      5167
Issue_Date          5167
Violation_Time      5167
Violation           5167
Fine_Amount         5167
Penalty_Amount      5167
Interest_Amount     5167
Reduction_Amount    5167
Payment_Amount      5167
Amount_Due          5167
County              5167
Issuing_Agency      5167
dtype: int64
>>> violation_data2 = violation_data2[violation_data2.Issue_Date <= ts]
>>> violation_data2.loc[violation_data2.Issue_Date>ts,:].count()
Plate               0
State               0
License_Type        0
Summons_Number      0
Issue_Date          0
Violation_Time      0
Violation           0
Fine_Amount         0
Penalty_Amount      0
Interest_Amount     0
Reduction_Amount    0
```

Figure 9: Before removing the data until the year 2017

After deleting records with year greater than 2017, we had a total of 993948 records.

```
>>> violation_data2.count()
Plate               993984
State               993984
License_Type        993984
Summons_Number      993984
Issue_Date          993984
Violation_Time      993984
Violation           993984
Fine_Amount         993984
Penalty_Amount      993984
Interest_Amount     993984
Reduction_Amount    993984
Payment_Amount      993984
Amount_Due          993984
County              993984
Issuing_Agency      993984
dtype: int64
```

Figure 10: After removing the data

The Violation_Time column had both incomplete and incorrect entries, we eliminated those individual entries to avoid ambiguity. The final count of records after the entire Data Cleaning was 993976.

```
>>> #Eliminating Erroneous data for Violation time
... violation_data1 = violation_data1[violation_data1.Violation_Time != '2:30']
...
>>> violation_data1 = violation_data1[violation_data1.Violation_Time != '4:37']
>>> violation_data1 = violation_data1[violation_data1.Violation_Time != '6:00']
>>> violation_data1 = violation_data1[violation_data1.Violation_Time != '37:30PM']
>>> violation_data1 = violation_data1[violation_data1.Violation_Time != '14:36PM']
>>> violation_data1 = violation_data1[violation_data1.Violation_Time != '18:59PM']
>>> violation_data1 = violation_data1[violation_data1.Violation_Time != '14:41PM']
>>> violation_data1 = violation_data1[violation_data1.Violation_Time != '08:+0AM']
>>> violation_data4.count()
Plate               993976
State               993976
License_Type        993976
Summons_Number      993976
Issue_Date          993976
Violation_Time      993976
Violation           993976
Fine_Amount         993976
Penalty_Amount      993976
Interest_Amount     993976
Reduction_Amount    993976
Payment_Amount      993976
Amount_Due          993976
County              993976
Issuing_Agency      993976
dtype: int64
```

Figure 11: Count after cleaning the dataset

Adjustments to data types:

Adjustments were made to the column data types after careful investigation of the columns. Below screenshots explain how the data types were modified.

```
... violation_data4['Fine_Amount']=violation_data4['Fine_Amount'].astype(float)
...
>>> violation_data4['Fine_Amount']=violation_data4['Fine_Amount'].astype(int)
>>> violation_data4['Interest_Amount']=violation_data4['Interest_Amount'].astype(float)
>>> violation_data4['Interest_Amount']=violation_data4['Interest_Amount'].astype(int)
>>> violation_data4['Reduction_Amount']=violation_data4['Reduction_Amount'].astype(float)
>>> violation_data4['Reduction_Amount']=violation_data4['Reduction_Amount'].astype(int)
>>> violation_data4['Penalty_Amount']=violation_data4['Penalty_Amount'].astype(float)
>>> violation_data4['Penalty_Amount']=violation_data4['Penalty_Amount'].astype(int)
>>> violation_data4['Payment_Amount']=violation_data4['Payment_Amount'].astype(float)
>>> violation_data4['Payment_Amount']=violation_data4['Payment_Amount'].astype(int)
>>> violation_data4['Amount_Due']=violation_data4['Amount_Due'].astype(float)
>>> violation_data4['Amount_Due']=violation_data4['Amount_Due'].astype(int)
...
```

```
>>> violation_data4.dtypes
Plate                      object
State                      object
License_Type               object
Summons_Number              int64
Issue_Date          datetime64[ns]
Violation_Time             object
Violation                  object
Fine_Amount                 int32
Penalty_Amount              int32
Interest_Amount             int32
Reduction_Amount            int32
Payment_Amount              int32
Amount_Due                  int32
County                     object
Issuing_Agency             object
dtype: object
```

Figure 12: Adjustment of datatypes

Outliers:

After removing the Null values, we tried to find the range of dollars that has been charged with the violation committed. Here we are taking into consideration only the Fine_Amount and the Payment_Amount because all the other variables such as Penalty_Amount, Interest_Amount, and Reduction_Amount are not applicable to every row in the table. So even if we try to plot these three variables we either get a box plot with only outliers or just a clean boxplot without outliers. Therefore, we are considering only the Fine_Amount (that is being charged at the time

of the violation committed) and the Penalty_Amount (that is the final amount being paid by a person upon issuing a summon) in the current analysis.

Fine_Amount:

From the below screenshot of the boxplot of Fine_Amount, there are several data points that lie above the horizontal bar. These are the data points that are considered outliers for the Fine_Amount. We removed those outliers since we have 97% of the data between $25 and $180. We will, however, do a transformation on the data and check if there are any changes in the number of outliers.

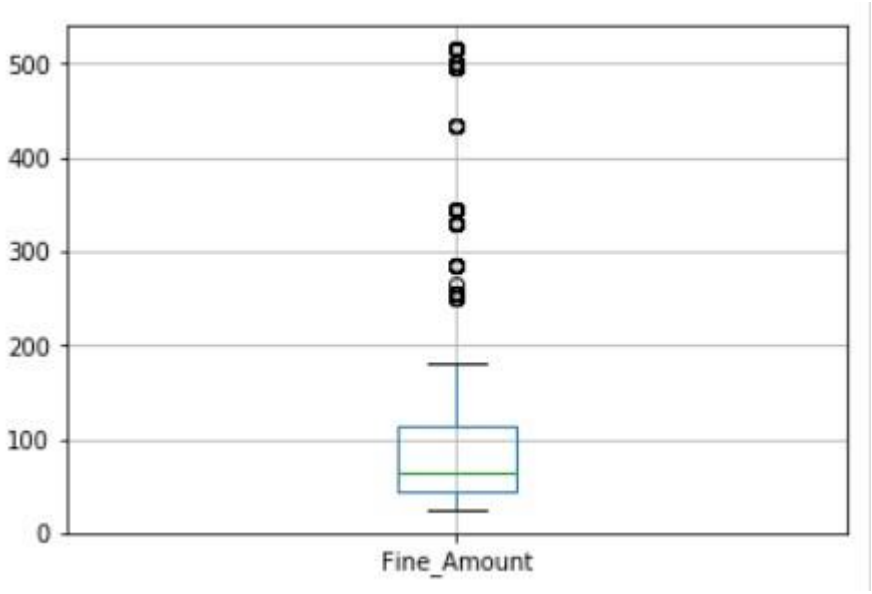

Figure 13: Outlier-Fine_Amount

Data Transformation:

Log Transformation:

The log transformation for the Fine_Amount is performed, and the below graph conveys us that the transformation did not help much. We can eliminate the outliers and proceed with other columns.

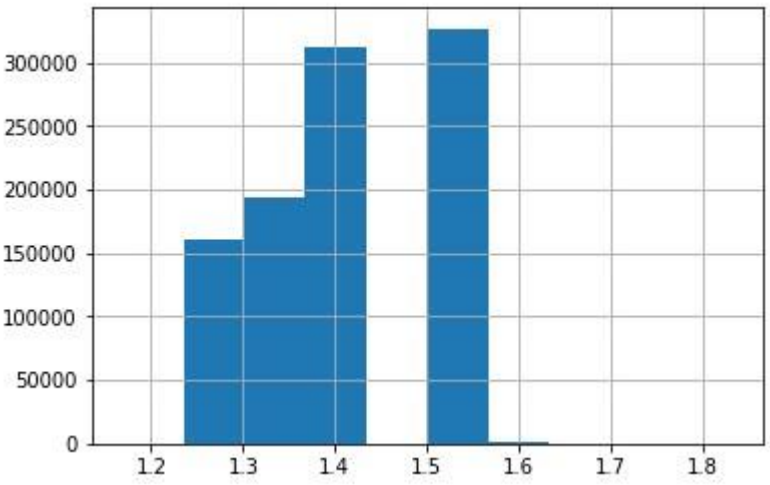

Figure 14: Log Transformation

Payment_Amount:

From the below box plot of Payment_Amount, we found there are several data points that lie above the horizontal bar. These are the data points that are considered outliers for the Payment_Amount. Since the number of outliers is significant, we do a transformation and check if there are any changes in the number of outliers.

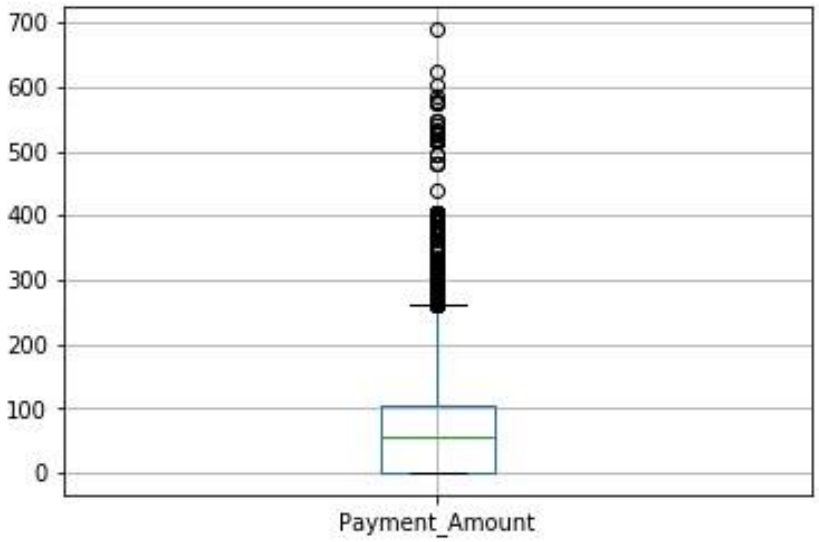

Figure 15: Outlier- Payment_Amount

The below graph shows the log transformation on Payment_Amount and it is not showing a normalized data, so we proceed by neglecting the data points above the horizontal bar.

```
violation_data1['Payment_Amount']=np.log(violation_data1['Payment_Amount'])
violation_data1['Payment_Amount'].apply(np.log).hist()
import matplotlib.pyplot as plt
plt.show()
```

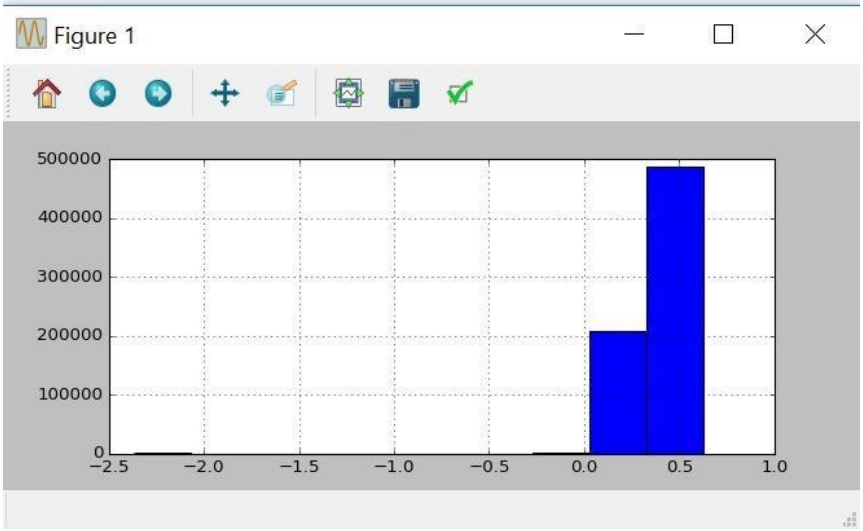

Figure 16:Box Plot - Payment_Amount

Data Reduction:

Principal Components Analysis (PCA):

PCA is a simple yet popular and useful linear transformation technique that is used in numerous applications to create a linear combination of columns. The main goal of a PCA analysis is to identify patterns in data; PCA aims to detect the correlation between variables. Upon finding a strong correlation among variables, we attempt to reduce the dimensions. Therefore, we find the variables that are responsible for causing maximum variation and try to project them in a smaller subspace while returning the important information.

```
In [45]: violation_data_pca = violation_data[['Fine_Amount','Penalty_Amount','Interest_Amount','Reduction_Amount','Payment_Amount','Amount_Due']]

In [46]: pca_result = pca(n_components=6).fit(violation_data_pca)

In [47]: pca_result.explained_variance_
Out [47]:
array([  6.33076311e+03,   2.17353975e+03,   9.90020380e+02,
         4.12293894e+02,   9.57094138e+01,   8.56368212e-01])

In [48]: pca_result.components_
Out [48]:
array([[-0.05898531,  0.20387901,  0.22848506, -0.02243699, -0.44402784,
         0.83969311],
       [ 0.44139311,  0.22311904,  0.12701372, -0.28020902,  0.74371816,
         0.32806058],
       [ 0.73456912, -0.20762701, -0.01522448,  0.62588466, -0.15362902,
         0.04164087],
       [-0.22794416,  0.760298  ,  0.07338234,  0.56924541,  0.16194725,
        -0.11973385],
       [-0.21005185, -0.34753622,  0.86953849,  0.20198065,  0.18471472,
        -0.06390534],
       [-0.40745643, -0.40829866, -0.41224768,  0.40547671,  0.40715386,
         0.40882477]])
```

Figure 17: PCA

From the above screenshot of eigenvalues, except the last value, all the others have a value greater than 1. This tells us that, we are dealing with only 5 variables out of the 6 variables. We cross-check this with the scree plot.

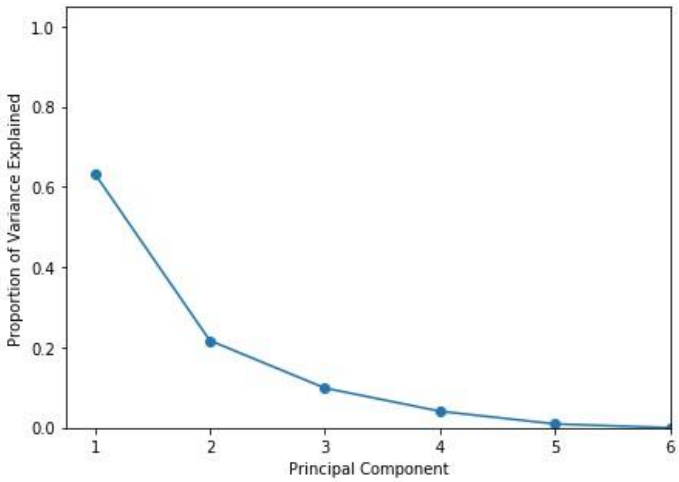

Figure 18: PCA 1

The scree plot above shows that after the component 5 it becomes flat or at least from 4 it is flat. This means that we are good with the PCA analysis and out of 6 variables we should be using only

5. Even though this is not much of a reduction in the variables it is still good to analyze them and decide if we must move with FA or not.

Descriptive Statistics:

It is always advised to study the variables in the data frame before performing any analysis on that. Here we have both numeric and object variables and we find the summary of them separately.

Categorical variables:

We have a total of eight columns that are categorical. Below screenshot provides the detailed summary of each.

```
In [30]: violation_data.describe(include=['object'])
Out [30]:
          Plate   State License_Type  Issue_Date Violation_Time  \
count    993976  993976       993976      993976         993976
unique   649034      65           69        3560           1503
top     47603MD      NY          PAS  2017-05-11        08:36AM
freq        106  767292       759660        6656           3215

                        Violation  County Issuing_Agency
count                      993976  993976         993976
unique                         87       9             13
top     NO PARKING-STREET CLEANING      NY        TRAFFIC
freq                       169794  318240         941707
```

Figure 19: Summary-Categorical

- Most open parking and camera violations occurred in the morning time when there is more traffic. There were 3215 instances of violations at the exact 8:36 AM.
- 11th May recorded the highest number of tickets issued.

- "No Parking - Street cleaning" tops the list on the type of violation with a staggering frequency of 169794.
- New York state has the maximum number of violations.
- A vehicle with the license plate '47603MD' has the most number of violation tickets.

Numerical variables:

We have a total of seven columns that are numerical. Below screenshot provides the detailed summary of each.

```
In [33]: round(violation_data.describe(include=['number']),2)
Out [33]:
       Unnamed: 0  Summons_Number  Fine_Amount  Penalty_Amount  \
count   993976.00    9.939760e+05    993976.00       993976.00
mean    522573.55    7.939977e+09        72.26           19.62
std     302638.22    1.231295e+09        31.71           25.77
min          0.00    1.038638e+09        25.00            0.00
25%     260263.75    7.373254e+09        45.00            0.00
50%     523248.00    8.487362e+09        65.00            0.00
75%     783885.25    8.526678e+09       115.00           60.00
max    1048571.00    8.588530e+09       515.00           60.00

       Interest_Amount  Reduction_Amount  Payment_Amount  Amount_Due
count        993976.00         993976.00       993976.00   993976.00
mean              8.79              9.93           58.19       32.54
std              20.99             26.44           49.88       68.60
min               0.00              0.00            0.00        0.00
25%               0.00              0.00            0.00        0.00
50%               0.00              0.00           55.00        0.00
75%               0.00              0.00          105.00        0.00
max             289.00            718.00          689.00      784.00
```

Figure 20: Summary-Numerical

- The average fine amount that a violator received was $72.26.
- The maximum amount that a violator received is $515.
- Maximum amount due that accumulated for a violator in 17 years is $784.

Below are the box plots and QQ plots for the continuous predictor variables:

Fine_Amount

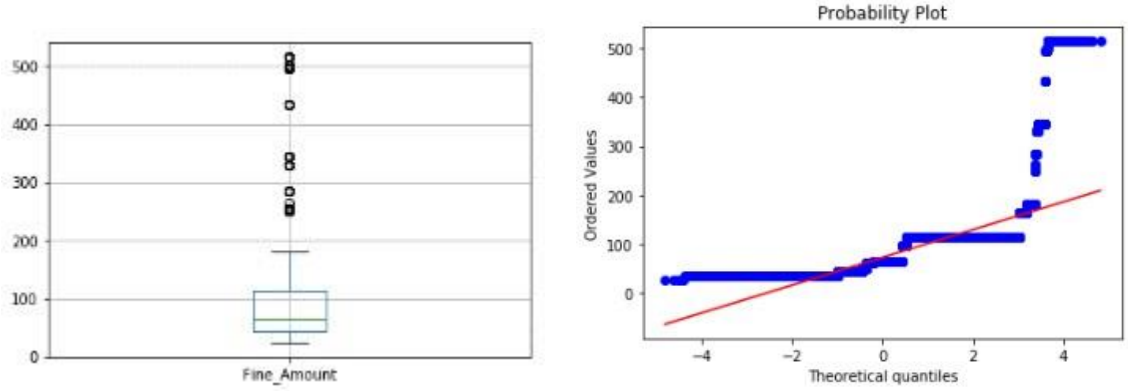

Figure 21: Descriptive-Fine_Amount

- In the above Box Plot of Fine_Amount, we can see there is a good number of Outliers above the Horizontal Bar.
- From the above QQ-plot of the Fine_Amount variable, we see the data points are not so properly represented on the standard reference indicating poor normality.

Penalty_Amount

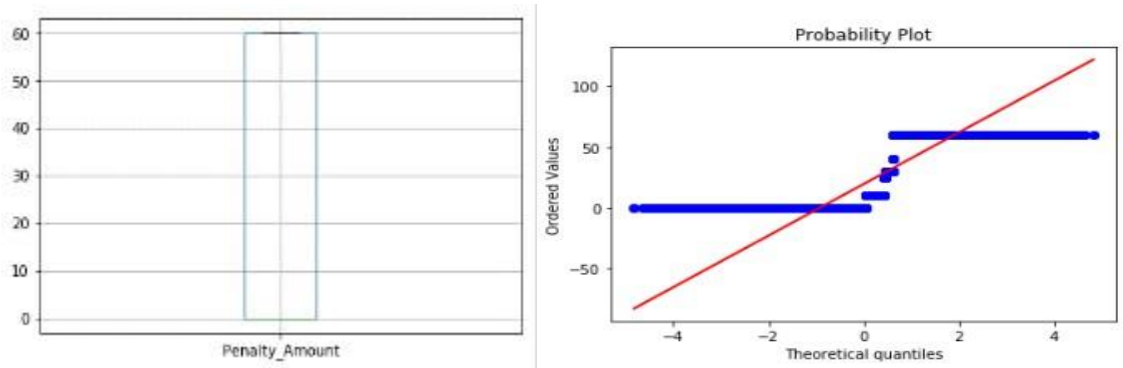

Figure 22: Descriptive-Penalty_Amount

- In the above Box Plot of Penalty_Amount, we can see there are no data points anywhere in the box plot.
- From the above QQ-plot of the Fine_Amount variable, we see the data points are not properly represented on the standard reference indicating very poor normality.

Reduction_Amount

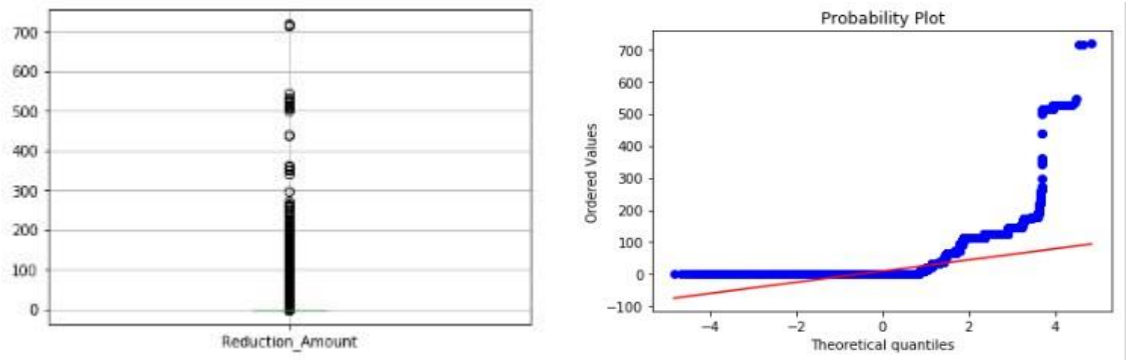

Figure 23-Reduction_Amount

- In the above Box Plot of Reduction_Amount we can see there are a good number of Outliers above the Horizontal Bar and a very few below the Horizontal bar.
- From the above QQ-plot of the Reduction_Amount variable, we see the data points are not so properly represented on the standard reference indicating weak normality.

Payment_Amount

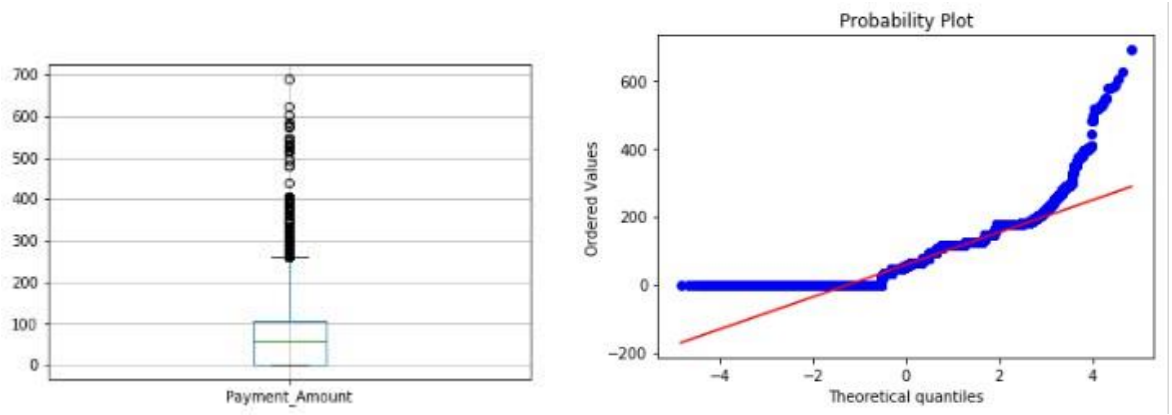

Figure 24:Descriptive-Payment_Amount

- In the above Box Plot of Payment_Amount, we can see there is again a good number of Outliers above the Horizontal Bar.
- From the above QQ-plot of the Payment_Amount variable, we see only a few data points are properly represented on the standard reference indicating weak normality.

Amount_Due

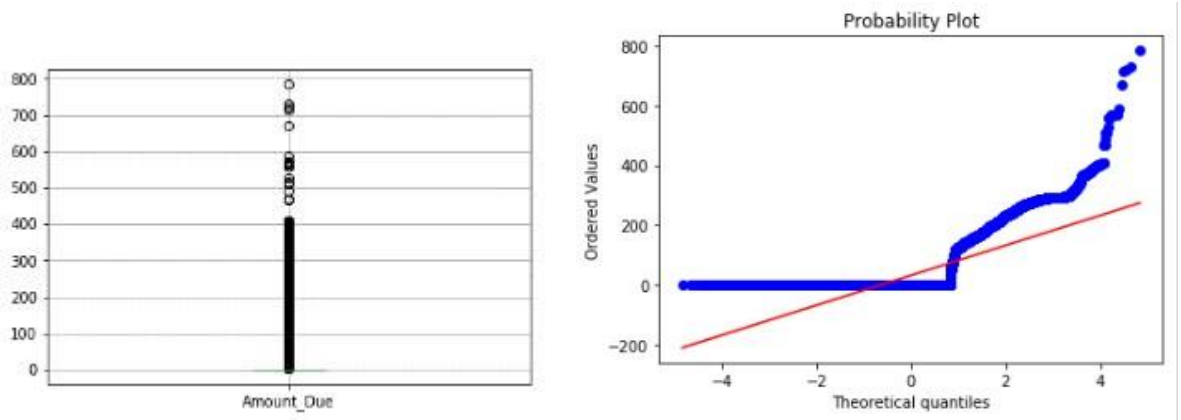

Figure 25: Amount_Due

- In the above Box Plot of Amount_Due, we can see all the Outliers are above the Horizontal Bar itself.
- From the above QQ-plot of the Amount_Due variable, we see the data points are not properly represented on the standard reference indicating very poor normality.

From all the above plots, Since None of them shows normality, we will be performing a log transformation on these variables before proceeding with our modeling techniques.

For the data frame we are using target variable as categorical and the predictor variables as a combination of continuous variables, so we will be using Neural Network for the analysis. Furthermore, it is very difficult to create dummy variables and interpret the results. Decision Tree

is a better idea as it handles categorical variables easily; therefore, we would go with decision tree as the second probable model.

Data Dictionary:

After careful observation, we found our data had few columns that did not provide any significance to our Analysis. Hence, we decided to eliminate those columns. Following are the columns that were removed:

- Judgment Entry Date
- Precinct
- Violation Status
- Summons Image

```
>>> violation_data.columns.tolist
<bound method IndexOpsMixin.tolist of Index(['Plate', 'State', 'License Type', 'Summons Number', 'Issue Date',
       'Violation Time', 'Violation', 'Judgment Entry Date', 'Fine Amount',
       'Penalty Amount', 'Interest Amount', 'Reduction Amount',
       'Payment Amount', 'Amount Due', 'Precinct', 'County', 'Issuing Agency',
       'Violation Status'],
      dtype='object')>
>>> violation_data1.drop(['Judgment Entry Date','Precinct','Violation Status'],axis=1,inplace=True)
>>> violation_data1.columns.tolist
<bound method IndexOpsMixin.tolist of Index(['Plate', 'State', 'License Type', 'Summons Number', 'Issue Date',
       'Violation Time', 'Violation', 'Fine Amount', 'Penalty Amount',
       'Interest Amount', 'Reduction Amount', 'Payment Amount', 'Amount Due',
       'County', 'Issuing Agency'],
      dtype='object')>
```

Figure 26: Dropping columns

After dropping unnecessary columns, we ended up with 15 columns that are of primary focus in our analysis and below are those columns with their corresponding details.

Attribute Name	Description	Data Type	Source
Plate	Vehicle Plate Number	Object	https://opendata.cityofnewyork.us/
State	State where the Violation happened	Object	https://opendata.cityofnewyork.us/
License_Type	Vehicle Registration Type	Object	https://opendata.cityofnewyork.us/
Summons_Number	Ticket Violation Number	Int	https://opendata.cityofnewyork.us/
Issue_Date	Date on which the ticket has been issued	Object	https://opendata.cityofnewyork.us/
Violation_Time	Time in which the Violation occurred	Object	https://opendata.cityofnewyork.us/
Violation	Type of Violation	Object	https://opendata.cityofnewyork.us/
Judgment_Entry_Date	Court order date	Date	https://opendata.cityofnewyork.us/
Fine_Amount	Fine Amount to be paid	Int	https://opendata.cityofnewyork.us/
Penalty_Amount	Penalty Amount	Int	https://opendata.cityofnewyork.us/
Interest_Amount	Interest Amount on fine	Int	https://opendata.cityofnewyork.us/
Reduction_Amount	Reduction Amount	Int	https://opendata.cityofnewyork.us/
Payment_Amount	Payment Amount	Int	https://opendata.cityofnewyork.us/
Amount_Due	Amount Due	Int	https://opendata.cityofnewyork.us/
County	County where the Violation happened	Object	https://opendata.cityofnewyork.us/
Issuing_Agency	Issuing department	Object	https://opendata.cityofnewyork.us/

Figure 27: Data dictionary

Modeling Techniques:

The aim of our analysis is to find the factors that affect or the factors that contribute to the violation that is happening. Here our target variable is the violation, which explains the violation

type. The variable violation a categorical variable with 87 categories. Since 87 categories were too much to handle, we mapped all the 87 categories into six main categories namely traffic, plate, meter, no standing, parking and other. We came up with the six broad classifications in the following ways:

- We combined all the violation categories that correspond to some form of a parking violation in the category “parking”.
- We combined categories like muni-meter, expired meter and few other meters related violations into the category “meter”.
- Violations with respect to no standing day time limit, no standing bus stop etc., are grouped in the “no standing” category.
- Expire plate violation, missing plate issues and other plate related issues are combined under the category called “plate”.
- The rest are merged into a category called “other”. This other category also contains such categories, which has fewer entries.

Since the chosen target variable and the independent variables contain more than two categories, the use of regression is ruled out by default. Therefore, we have decided to use decision trees, random forest, and neural networks to find the accuracy of the model we chose.

Decision tree

Decision tree generally uses the divide and conquer method to represent data. It is preferred over other methods because it provides a clear model explaining how the decision is made or

how the decision can be made. Each internal node of the decision tree corresponds to an attribute and each leaf node represents a class label.

Random Forest

Random forest algorithm is used for both classification and regression. As the name suggests it creates a forest with several random trees. With the increase in the number of trees, the accuracy also increases. To model the tree, we use the past data. Therefore, once you pass it to the decision tree, the tree will start building the chances of the variable to be predicted.

Neural Networks

Neural networks can be used to solve interesting problems once they are trained. They are very good at pattern recognition and non-linear classification problems. With the required elements also called as neurons we can classify our data with arbitrary accuracy. They are particularly well suited for multivariate data type classification problems like the one in our dataset and hence we have chosen neural networks as our third algorithm to make predictions.

Model Assumption

Decision tree

Since the target variable we chose is categorical, we are dealing with classification trees. In our model, the categorical variable is a violation, which has six categories. We assume that the independent variables fine amount and payment amount has some influence in deciding the violation that is happening. In addition, the tree performance is not be affected by the non-linear relationship between the variables. The third assumption is that it will be easy to interpret as

decision trees are very intuitive and easy to explain. We will now be able to find the combined effect of both final amount and payment amount on the violation. Finally, decision tree requires less data preparation efforts because the structure of the tree will stay the same.

Random Forest

Random forest is an ensemble method for classification and regression. With each individual tree built, it is said that we find the best split of data. When a tree is built, the complete dataset in never used, just a sample of it. Averaging is performed on the individual tree to get the aggregate. For the random forest, we use bootstrap aggregation, and this is the difference in between decision tree and random forest. Sampling is representative is the assumption for this.

Neural Networks

We do not have assumptions about data or errors. Theoretically, a neural network can approximate any function, and this is done without assumptions, it only depends on the data and network configuration.

Data Splitting and Subsampling

Our Target variable is “Violation” with 993975 records and distribution of this is as follows:

Violation type	Count
meter	233928
no standing	179081
parking	365124
plate	134203
traffic	50378
other	31261

Count of parking violation is higher as compared to other violation types. A decision tree can fit into a training dataset. The most important variables within the dataset are the top nodes on which tree split is done.

We use data splitting generally to have a robust predictive model. Data is split usually into 2 portions namely Training data and Test data. Sometimes even validation data is also considered. We also must split the data in a way that it should prevent overfitting and underfitting. If our model is overfitting then it generally unable to provide accurate predictions and if our model is under fitted then it would miss trends and data. Hence, data splitting is one the important aspects. Cross-validation is also one of the techniques that can be used to prevent this. We have split the data in the 75-25 ratio for the following reasons:

- 75% data is used as training data and 25% as testing data. This implies that our neural network is trained sufficiently and is a good contender for the robust predicter model.
- Since there are close to one million records, this split would be better for our model
- If we would have taken 50-50 or 60-40, our model would have missed certain data trends and points. That would have made this model as under fit and could not be used for identifying violation type and 80-20 would still make it non-robust model.

Comparison of variables between test and training data

Numerical Variables Comparison

The below summary of the numerical variables shows not much difference between the mean, median, standard deviation for the training and test datasets.

Training Data screenshot:

```
>>> X_train_summary.describe()
         Fine_Amount  Payment_Amount
count  575469.000000   575469.000000
mean       71.205778       60.791686
std        30.777690       48.811993
min        35.000000        0.000000
25%        45.000000        0.000000
50%        65.000000       60.000000
75%       115.000000      105.000000
max       515.000000      547.000000
```

Figure 28: Training dataset - Numerical

Testing Data screenshot:

```
>>> X_test_summary.describe()
         Fine_Amount  Payment_Amount
count  191823.000000   191823.000000
mean       71.268930       60.811743
std        30.799548       48.826756
min        25.000000        0.000000
25%        45.000000        0.000000
50%        65.000000       60.000000
75%       115.000000      105.000000
max       515.000000      518.000000
```

Figure 29: Testing dataset - numerical

Categorical Variables Comparison

The below summary of the categorical variables shows similarities in both the training and test data set, i.e., the frequent observation is ‘parking’. The percentage of the occurrence of the ‘parking’ observation is close to 35% in both training and test dataset.

Training Data screenshot

```
>>> y_test_summary.describe()
       Violation
count     191823
unique         6
top      parking
freq       67364
```

Figure 30: Training dataset - categorical

Testing Data screenshot

```
>>> y_train_summary.describe()
       Violation
count     575469
unique         6
top      parking
freq      201957
```

Figure 31: Testing dataset - categorical

Model Goals

Decision Tree Model

The decision tree is identified as the simplest model as it is very easy to interpret. The aim is to build a model that predicts the value of a target variable "violation". To encapsulate the training data in the smallest possible tree is a goal of the decision tree. Decision tree provides simple rules to split the node at the top of the tree, that is a root node. Various branches of different length are formed. Minimize the tree size is a logical rule so that decision can be made faster. Hence, we can easily interpret the effect of predictors "Fine Amount" and Payment Amount" on a target variable "violation".

Random Forest Model

Random forest is also popular model like the decision tree. This model is a good choice when the relationship between dependent and independent variable is non-linear. It provides ordered list of important variables and a majority tree is selected. Random forest runtimes are quite fast, so the performance of the overall model can be improved by this tree. Random forest combines tree with the notion of an ensemble. Prediction of a target variable "violation" would become easy with this model.

Neural Models

A neural network is nothing but an information processing paradigm that works in a similar way as how biological nervous system such as the brain. It is used to derive meaning from complex or imprecise data. It is used to find patterns and detect trends that are too complex to find by any other computing techniques or by a human. It mainly helps us arrive us to a solution for "what if" questions.

The neural network is comprised of an interconnected network of neurons and each has specific weights associated with it. Each neuron has specific weights connected to it along with independent variable values and a total bias factor is also considered. This network will help us predict our target variable "Violation"

Model Building

Decision Tree Model

While using a model that has both categorical variables and numerical variables in a decision tree, CART (Classification and Regression trees) procedure is usually chosen. This is because the parent node is divided into two child nodes, one that satisfies the parent condition and the other node

is when the parent condition is not satisfied. The child nodes are again divided in the same way.

The screenshot below shows the code used to generate the decision tree.

```
In [37]: from sklearn.tree import DecisionTreeClassifier
    ...: classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
    ...: classifier.fit(X_train, y_train)
    ...:
    ...: # Predicting the Test set results
    ...: y_pred = classifier.predict(X_test)
    ...:
    ...: # Making the Confusion Matrix
    ...: from sklearn.metrics import confusion_matrix
    ...: cm = confusion_matrix(y_test, y_pred)
    ...:
    ...: from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
    ...: accuracy_score(y_test,y_pred)
    ...: classification_report(y_test,y_pred)
    ...:
Out[37]: '             precision    recall  f1-score   support\n\n      meter       0.98      0.76      0.86     58657\nno standing       0.56
   0.93      0.69     44795\n      other       0.79      0.01      0.03      7782\n    parking       0.92      0.65      0.76     91241\n      pl
ate       0.57      0.96      0.72     33425\n    traffic       1.00      0.63      0.77     12594\n\navg / total       0.82      0.75      0.74
   248494\n'

In [38]: accuracy_score(y_test,y_pred)
Out[38]: 0.74575643677513337
```

```
In [62]: from sklearn.metrics import confusion_matrix
    ...: cm = confusion_matrix(y_test, y_pred)
    ...:

In [63]: print(cm)
[[44676     4     1   123 13853     0]
 [  152 41525    20  2128   965     5]
 [  388  3877   108  2047  1361     1]
 [   33 24328    10 59020  7846     4]
 [  408   773     0   179 32065     0]
 [   10  4234     2   386    32  7930]]
```

Figure 32: Confusion Matrix - Decision tree

Decision tree model has 74.57% accuracy. Since the tree generated was too big, we had to prune the tree to analyze it further. The code for the prune tree is given below. Following the screenshot is the tree generated diagram.

```
In [33]: col_names = list(violation_data.ix[:,(7,11)].columns.values)
    ...: classnames = list(violation_data.Violation.unique())
    ...:
/Users/mrunal/anaconda3/bin/ipython:1: DeprecationWarning:
.ix is deprecated. Please use
.loc for label based indexing or
.iloc for positional indexing

See the documentation here:
http://pandas.pydata.org/pandas-docs/stable/indexing.html#ix-indexer-is-deprecated
  #!/Users/mrunal/anaconda3/bin/python

In [34]: tre4 = tree.DecisionTreeClassifier(min_samples_split=210,min_samples_leaf=210)
    ...: tre4.fit(violation_data.ix[:,(7,11)],violation_data.Violation)
    ...: violation_data = StringIO()
    ...: tree.export_graphviz(tre4, out_file=violation_data,
    ...:                      feature_names=col_names,
    ...:                      class_names=classnames,
    ...:                      filled=True,
    ...:                      rounded=True,
    ...:                      special_characters=True)
    ...: graph = pydotplus.graph_from_dot_data(violation_data.getvalue())
    ...: graph.write_pdf('tree1.pdf')
    ...:
Out[34]: True
```

Figure 33: Decision tree

```
>>> cv_scores = cross_val_score(classifier, X, y, cv=5)
>>> print(cv_scores)
[ 0.72031982  0.77919979  0.77879941  0.77245743  0.78752867]
>>> print("Average 5-Fold CV Score: {}".format(np.mean(cv_scores)))
Average 5-Fold CV Score: 0.7676610253694582
```

Figure 34:Decision tree - cross validation

We selected k fold cross-validation technique to assess the predictive models. We chose 5-fold cross-validation to prevent underfitting and overfitting. The average accuracy of the model we received is 76.76%.

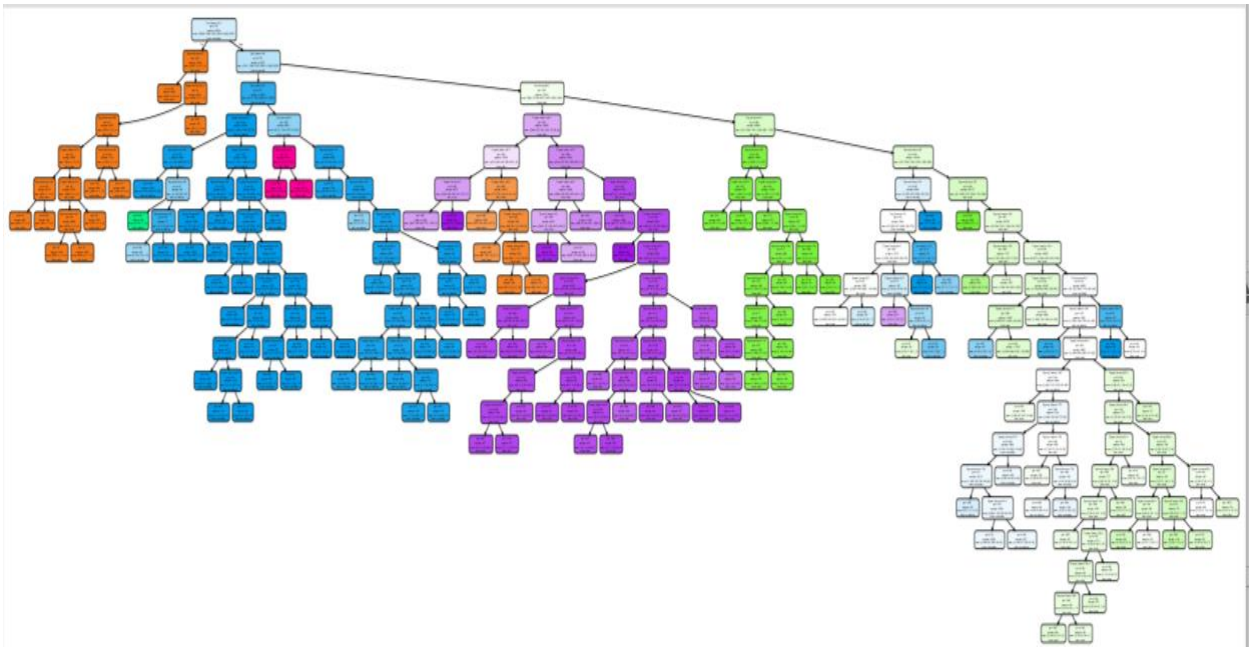

Figure 35: Decision tree output

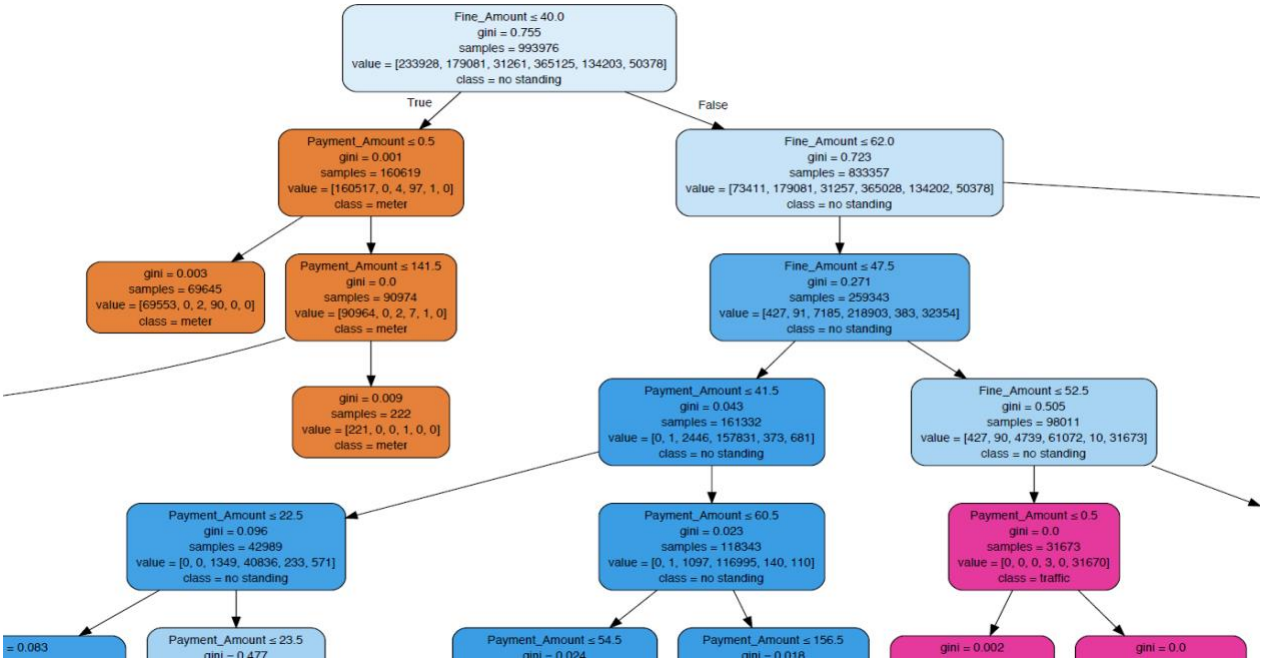

Figure 36: Prune tree

Results and Interpretation

From the above screenshot of the decision tree, we can see that the important factor or the variable that influences the violation is the fine_amount. So, since a higher number of violations are in the range of $40 it is taken as the comparing factor to decide the violation. The payment_amount which is the sum of all the penalty that has been charged with a violation has the least influence in the violation.

Here, with respect to the CART procedure, we can say that if the Fine_Amount is less than $40 it is considered as a YES and if the Fine_Amount is greater than $40 it is considered as NO. So, if the Fine_Amount condition is YES, irrespective of the Payment_Amount, the violation is of type METER. If the Fine_Amount is greater than $40 but less than $62, we have a broader classification of violation.

If the fine_amount is greater than $40 but less than $47.5 and if the Payment_amount is less than $22.5 the violation is charged under NO Standing violation. If the Fine_Amount is $47.5 and the Payment_amount is greater than $22.5 we have both Parking violation and No Standing violation.

From the analysis, we observed one thing, that the parking violation, which has the highest count is charged between $40 and $100 whereas the plate violation, very few are charged as high as $270 minimum.

Random Forest Model

```
In [39]: from sklearn.ensemble import RandomForestClassifier
    ...: classifier = RandomForestClassifier(n_estimators=10, criterion = 'entropy', random_state = 0)
    ...: classifier.fit(X_train, y_train)
    ...:
Out[39]:
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='entropy',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=0, verbose=0, warm_start=False)

In [40]: # Predicting the Test set results
    ...: y_pred = classifier.predict(X_test)
    ...:
    ...: # Making the Confusion Matrix
    ...: from sklearn.metrics import confusion_matrix
    ...: cm = confusion_matrix(y_test, y_pred)
    ...:
    ...: from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
    ...: accuracy_score(y_test,y_pred)
    ...:
Out[40]: 0.74578058222733745


In [41]: from sklearn.ensemble import RandomForestClassifier
    ...: classifier = RandomForestClassifier(n_estimators=100, criterion = 'entropy', random_state = 0)
    ...: classifier.fit(X_train, y_train)
    ...:
    ...: # Predicting the Test set results
    ...: y_pred = classifier.predict(X_test)
    ...:
    ...: # Making the Confusion Matrix
    ...: from sklearn.metrics import confusion_matrix
    ...: cm = confusion_matrix(y_test, y_pred)
    ...:
    ...: from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
    ...: accuracy_score(y_test,y_pred)
    ...:


Out[41]: 0.74578863071140555
```

Figure 37: Random forest

After decision tree, we ran Random Forest model and found 74.57 % accuracy which is like decision tree accuracy. It will not help us much, so we will run Neural model for further analysis.

```
...
>>> # Compute 5-fold cross-validation scores: cv_scores
... cv_scores = cross_val_score(classifier, X, y, cv=5)
...
>>> # Print the 5-fold cross-validation scores
... print(cv_scores)
...
[ 0.72037195  0.77906295  0.77885806  0.77250956  0.78742441]
>>> print("Average 5-Fold CV Score: {}".format(np.mean(cv_scores)))
Average 5-Fold CV Score: 0.7676453858601441
>>>
```

Figure 38: Random forest -cross-validation

Neural Models

We have considered 2 independent variables i.e. Fine_Amount and Payment_Amount as input layers for our model. Also, we have built neural networks using 2 widely used modeling techniques namely logistic and relu. Since our output variable is categorical, we have used classifier. Hidden layer has been kept as 100 and hidden size is kept as default i.e. 100. We will be implementing two types of neural model and comparing one to another.

Logistic Neural Model

```
MLPClassifier(activation='logistic', alpha=0.0001, batch_size='auto',
       beta_1=0.9, beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100, 100), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='sgd', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
>>> nnclass3_pred = nnclass3.predict(X_test)
>>> cm = metrics.confusion_matrix(y_test, nnclass3_pred)
>>> print(cm)
[[31148     1     0   100 13332     0]
 [    0 23027     0  7672   864     0]
 [    2  2149     0  2424   612     1]
 [   17 12600     0 49358  5389     0]
 [    1   334     0   452 32289     0]
 [    0  2671     0   874     1  6505]]
```

Figure 39: Logistic Neural model

- Activation type here is provided as logistic. One of the main reasons why we choose neural logistic model is because this model is good our input dimension data is very high, and our data is not linearly separable.

```
>>> accuracy_score(y_test,nnclass3_pred)
0.74197046235331532
>>> print(classification_report(y_test,nnclass3_pred))
             precision    recall  f1-score   support

      meter       1.00      0.70      0.82     44581
no standing       0.56      0.73      0.64     31563
      other       0.00      0.00      0.00      5188
    parking       0.81      0.73      0.77     67364
      plate       0.62      0.98      0.75     33076
    traffic       1.00      0.65      0.79     10051

avg / total       0.77      0.74      0.74    191823
```

Figure 40: Accuracy - Logistic neural model

Our confusion matrix placed above tell us that how our model has performed.

- We have very good precision for the meter, parking, and traffic. For others a violation-type we have got no precision.
- For violation as no standing and plate, we have got a medium precision which is not bad.
- Our model has a total of 74% accuracy.
- Misclassification rate is calculated by false positive and false negative by total and is 25.09%.

Relu Neural Model

Next, we chose to run relu neural model as it is one of the most widely used. One of the main advantages of this model is it takes less time for training data to run unlike logistic which took almost doubt to run. The main reason why relu takes lesser time is it does not involve computation for the exponential function. This seemed like the ideal model because we have a very large dataset.

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(100, 100), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='sgd', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
>>> nnclass2_pred = nnclass2.predict(X_test)
>>> cm = metrics.confusion_matrix(y_test, nnclass2_pred)
>>> print(cm)
[[35319     7     0    72  9183     0]
 [  151 27703     0  2989   720     0]
 [   92  2855    14  1704   522     1]
 [   30 16323    11 45621  5379     0]
 [  593   569     0   216 31698     0]
 [    0  3082     0   463     1  6505]]
>>> print(classification_report(y_test,nnclass2_pred))
             precision    recall  f1-score   support

      meter       0.98      0.79      0.87     44581
no standing       0.55      0.88      0.67     31563
      other       0.56      0.00      0.01      5188
    parking       0.89      0.68      0.77     67364
      plate       0.67      0.96      0.79     33076
    traffic       1.00      0.65      0.79     10051

avg / total       0.81      0.77      0.76    191823
```

Figure 41: Relu Neural Model

Our confusion matrix placed above tell us that how our model has performed:

- Violation type as meter, parking, plate, and traffic have performed well in terms of accuracy.
- Violation type as other has given us a precision 56% which is much better than no precision in the previous model.
- No-standing has performed decently with 56% accuracy.
- Our neural model with activation as relu has given us a total of 76% accuracy.
- Misclassification rate is calculated by false positive and false negative by total and is 23.43%.

Model Assessments

Decision Tree Model

Accuracy

```
In [41]: from sklearn.ensemble import RandomForestClassifier
    ...: classifier = RandomForestClassifier(n_estimators=100, criterion = 'entropy', random_state = 0)
    ...: classifier.fit(X_train, y_train)
    ...:
    ...: # Predicting the Test set results
    ...: y_pred = classifier.predict(X_test)
    ...:
    ...: # Making the Confusion Matrix
    ...: from sklearn.metrics import confusion_matrix
    ...: cm = confusion_matrix(y_test, y_pred)
    ...:
    ...: from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
    ...: accuracy_score(y_test,y_pred)
    ...:


Out[41]: 0.74578863071140555
```

Figure 42:Decision tree - accuracy

Decision tree model has total of 74.57% accuracy.

ROC Curve

ROC or Receiving operating characteristic is used to test to the performance of a binary classifier. It depicts how good the classifiers explain the true positives and at what level it shows the false positives. So, it defines the rate at which a prediction is performed correctly.

ROC is said to give perfect results when the classifier is binary. In our model since the classifier is not binary, it is difficult to calculate and interpret the results. Therefore, we have not used ROC curve to study the truth of the prediction results.

Strengths

- The accuracy of the decision tree is high. So here we can say that prediction accuracy of target variable violation is high.

- The big advantage of using decision tree is that it is very easy to understand and interpret. Also, the visual display is good. So, a person not having much analytics knowledge can also understand and interpret the model.
- Linearity is not required in the decision tree execution. By using decision tree model, we can expect the correct result for the nonlinear model as well.
- Data preparation efforts can be reduced by using a decision tree.
- Missing values do not affect the decision tree splitting and result.
- Outliers will not make any difference in decision tree analysis.
- Continuous and target variables can be handled by a decision tree.

Weakness

- Prediction of continuous target variables is challenging task in case of the decision tree.
- Decision trees have dynamic behavior so it might create issues.
- The analysis might be difficult for large and complex decision tree. Knowledge of tree pruning is required in this case.
- The relationship between target variable and predictors might change in case of the decision tree. Because any minor change in data can affect result so it could affect the analysis.

Neural Network Model

Misclassification Rate

The misclassification error refers to the number of an individual that we know that belong to a category that is classified by the method in a different category. Misclassification rate is calculated by the sum of false positive and false negatives divided by the total.

Our misclassification rate for the logistic neural model is 25.09% and misclassification rate for relu neural model is 23.43%. Relu model performed better in terms of misclassification rate. Also, another thing to note is for the violation type as 'other' logistic neural model failed miserably whereas relu model performed well for this as well. Hence from misclassification rate, we should be choosing relu over the logistic neural model

Accuracy

Classification accuracy is the ratio of correct predictions to total predictions made. Our accuracy for the logistic neural model is 74.09 whereas relu neural model has 76.56. Hence our neural network performs better.

ROC Curve

ROC is said to give perfect results when the classifier is binary. In our model since the classifier is not binary, it is difficult to calculate and interpret the results. Therefore, we have not ROC curve to study the truth of the prediction results.

Strengths

- Neural networks can be trained according to our training data. Hence it has the capability to become a good model for any type of categorical and numerical data set. For our data set consisting of categorical data, we could not use logistic regression directly and we found this a great model to work on.
- The range of problems that can be used for a neural network is huge. From text recognition to time series, it caters to different types of data to solve different statistical models.
- Different library implementations are available. With just a single parameter change, we can run several types of neural model. For our scenario, we have used logistic and relu.

- Neural networks are flexible. It depends on architect choice, the way they want to train and the algorithm they want to use.

Weakness

- Neural networks are hard to tune because it is equivalent to the black box. It depends heavily on initial parameters i.e. the starting point for gradient descent when training backpropagation networks. In the same way, they are hard to debug. It is quite difficult to understand how neural networks work.
- These are not suited for all types of problems. So, sometimes while trying to build the model for which they are not suited gives us the incorrect result. Neural networks are better suited for the specific type of problems. Hence, if the problems are vague, it results in an unstable model.
- Our neural network is not good for understanding the problem. It should be considered only when you have analyzed the data.

References

https://data.cityofnewyork.us/City-Government/Open-Parking-and-Camera- Violations/nc67

http://www.cs.princeton.edu/courses/archive/spr07/cos424/scribe_notes/0220.pdf

https://stats.stackexchange.com/questions/59124/random-forest-assumptions

https://stackoverflow.com/questions/38469925/normality-assumption-neural-networks

https://pdfs.semanticscholar.org/f3b6/f159116326209b1b4ea69e28db020f122ce0.pdf