CIS 4130

Machine Learning Semester Project

Lorenavasquez@baruchmail.cuny.edu

Milestone 1:

Due 9/9/2022 (10 points)

Proposal: Research and find a data set larger than 10 GB (should be free, open source etc.) OR describe a plan to collect at least 10 GB of data (e.g., from Twitter or other API). Some suggestions for data sets include Kaggle, DataHub.io, Data.gov, Open Data on AWS, UCI Machine Learning Repository, and Google Public Datasets. Try and pick a data set that aligns with your personal interests. Write up a brief 1 page proposal that includes a description of the data set, its attributes, and a description of what you intend to model, predict, forecast, etc. using the data set.

Here is the link to the dataset that I have chosen to work with.
https://data.cityofnewyork.us/City-Government/Open-Parking-and-Camera-Violations/nc67-uf89

I have chosen this dataset, as I recently received a traffic violation through the mail, that a camera had caught me in camera. Surprisingly, I haven't received one yet, but just in my city they have been installing these cameras all over the place! The data set that I have chosen is about 22GB.s large. If necessary, I will have to filter out the information to create it to a more suitable size, with the professor's advice.

Initially this dataset had been loaded onto the NYC OpenData database dated back to 2016. As of May 2016, the dataset "Open Parking and Camera Violations" is contained and updated continuously. As new violations are being issued, existing violations updated the dataset are also updated as well. Whether it is a new or an open violation the dataset is being updated weekly on Sunday. When a violation is satisfied then it has been paid for or dismissed via a hearing, statutorily expired or had other changes to its status, will be updated daily from the days Tuesdays through Sunday. Any violations that have been written off because of it being expired or no longer valid are indicated with blank financials. There is a summons column where we can visually see the summons, but the images will not be available on Sundays from 5am to 10am. The dataset is provided by data from the Department of Finance. Last known day to be updated, last day that I checked on the dataset on September 7th 2022.

In total there are 85.5M rows and 19 columns. Each row shows the open parking and camera violation that had been issued. The columns that I will be working with will be: (Plate, State, License Type, Issue Date, Violation Time, Violation, Fine Amount, Penalty Amount, Interest Amount, Reduction Amount, Payment Amount, Amount Due, Precinct, County, Issuing Agency, Violation Status and Summons Image).

What I will like to find:

1. **Out of the violations, what is the percentage of the violations that get written off?**
   As mentioned above - if a row's financial amounts are blank then the issuance has been written off because of it being expired or no longer valid.
2. **Is there a correlation between the license type?**
3. **Is there a correlation in the time that a violation is being issued?**
4. **What violations are most common, least common?**

5. **Is there a correlation with the counties and the violations they issue? Is there a prominent issue between a county and violations?**
6. **Is there a correlation with the issuing agency and the violations they issue?**
7. **Is there a correlation with the State Plate and the violations issued? Do non-residents have a tendency for a violation?**

Feedback from professor:

Hi Lorena:

I have reviewed your project proposal to work with the NYC Parking. My comments are as follows:

This is an interesting data set. You have pointed out a number of different correlations that can be examined. I am wondering if there is a model you can build to predict getting a ticket (or getting the fine waived etc). Building and testing a prediction model will make this a more interesting project. Spend a bit more time to come up with this last portion of the proposal and then resubmit.

Cheers, Prof. H.

Revised Proposed Plan:

To make the project more interesting, I will still do the 1-8 correlations between the variables that I mentioned. I would like to focus on building and testing a prediction model. Where I will predict getting a ticket based on the license plate (whether it is NJ or PA or NY) or the time of the violation. That way we can predict the type of violation based on the plates or on the time of the violation!

<u>Milestone 2</u>

Due 9/23/2022 9/30/2022(15 points)

Data Acquisition: Download or collect the data directly into a bucket on Amazon S3 (or in an AWS hosted database if that is more appropriate). Document the code, commands and steps you used to collect the data. If you are collecting data from an API, show the code used. If you are downloading the data from a site, document the commands used to download the data. The downloading process should be able to be automated (scripted) in code and repeatable. The data should not be downloaded to your own computer. Add a new section to your project document with all of the above details.

Configuring AWS CLI:

1. Log into AWS
2. Click on an EC2 instance and connect to it
    a. If you have the instance stopped from some previous work on it before, make sure to “Start” the instance once again. This can be done under the actions for instance.
3. Run the AWS CLI Configuration command: **aws configure**

4. At the prompt: AWS Access Key ID **paste in your Access Key ID**
5. At the prompt: AWS Secret Access Key **paste in your secret access key**
6. At the prompt: Default region name Type in: **us-east-2**
7. At the prompt: Default output format Type in: **json**
8. Test to see if AWS CLI is working by requesting the EC2 instance information: **aws ec2 describe-instances**
9. List buckets on Amazon S3: **aws s3 ls**
10. List IAM Users: **aws iam list-users**

Working with data in S3:

- We have already launched an EC2 instance with Amazon Linux AMI, and we have also configured the AWS CLI with our Access Key.
- <u>Next we need to create a bucket in S3:</u>

aws s3api create-bucket --bucket my-data-bucket-**XX** --region us-east-2 \
--create-bucket-configuration LocationConstraint=us-east-2

- <u>The XX can then be changed to my initials, so my command was:</u>

aws s3api create-bucket --bucket my-data-bucket-**LV2** --region us-east-2 \
--create-bucket-configuration LocationConstraint=us-east-2

```
[ec2-user@ip-172-31-46-142 ~]$ pip3 install boto3 pandas fsspec s3fs
Defaulting to user installation because normal site-packages is not writeable
Collecting boto3
  Downloading boto3-1.24.84-py3-none-any.whl (132 kB)
     |████████████████████████████████| 132 kB 4.2 MB/s
Collecting pandas
  Downloading pandas-1.3.5-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.3 MB)
     |████████████████████████████████| 11.3 MB 26.4 MB/s
Collecting fsspec
  Downloading fsspec-2022.8.2-py3-none-any.whl (140 kB)
     |████████████████████████████████| 140 kB 34.9 MB/s
Collecting s3fs
  Downloading s3fs-2022.8.2-py3-none-any.whl (27 kB)
Collecting s3transfer<0.7.0,>=0.6.0
  Downloading s3transfer-0.6.0-py3-none-any.whl (79 kB)
     |████████████████████████████████| 79 kB 13.4 MB/s
Collecting botocore<1.28.0,>=1.27.84
  Downloading botocore-1.27.84-py3-none-any.whl (9.2 MB)
     |████████████████████████████████| 9.2 MB 21.2 MB/s
```

Next we need to find a way that the data will be downloaded into the bucket via the website of the dataset.

```
>>> import pandas as pd
>>> df = pd.read_csv('s3://my-data-bucket-lv2/Open_Parking_and_Camera_Violations.csv')
```

<u>Milestone 2:</u>

- Once clicking on instances I then "restarted" my instance. (In cases where instances are not needed to be run, I stop the instance. **Actions > Start Instance / Stop Instance**.
- Now that the Instances are running, I then click on the checkbox to click on the instance and click on **Connect**.

```
       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
10 package(s) needed for security, out of 19 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-46-142 ~]$
```

- This screen is then prompted where I then run the command * sudo yum update

```
Updated:
  amazon-ssm-agent.x86_64 0:3.1.1732.0-1.amzn2          chrony.x86_64 0:4.2-5.amzn2.0.2
  dhclient.x86_64 12:4.2.5-79.amzn2.1.1                 dhcp-common.x86_64 12:4.2.5-79.amzn2.1.1
  dhcp-libs.x86_64 12:4.2.5-79.amzn2.1.1                ec2-net-utils.noarch 0:1.7.1-1.amzn2
  gnupg2.x86_64 0:2.0.22-5.amzn2.0.5                    initscripts.x86_64 0:9.49.47-1.amzn2.0.3
  kernel-tools.x86_64 0:5.10.144-127.601.amzn2          kpatch-runtime.noarch 0:0.9.4-6.amzn2
  libxml2.x86_64 0:2.9.1-6.amzn2.5.6                    libxml2-python.x86_64 0:2.9.1-6.amzn2.5.6
  microcode_ctl.x86_64 2:2.1-47.amzn2.0.13              systemd.x86_64 0:219-78.amzn2.0.20
  systemd-libs.x86_64 0:219-78.amzn2.0.20               systemd-sysv.x86_64 0:219-78.amzn2.0.20
  tzdata.noarch 0:2022d-1.amzn2.0.1                     zlib.x86_64 0:1.2.7-19.amzn2.0.2

Complete!
```

- Once completed, it will say “Complete!”. I will then do the AWS Configure - one more time to make sure it is configured.
- Now that I have my bucket on S3, the next step is to be able to use Python on this console. After being able to load in python, I will then be able to call on the API to retrieve the data from the website.

```
[ec2-user@ip-172-31-46-142 ~]$ pip3 install boto3 pandas fsspec s3fs
```

```
[ec2-user@ip-172-31-46-142 ~]$ python3 --version
Python 3.7.10
[ec2-user@ip-172-31-46-142 ~]$ python3
Python 3.7.10 (default, Jun  3 2021, 00:02:01)
[GCC 7.3.1 20180712 (Red Hat 7.3.1-13)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import boto3
>>> s3 = boto3.resource('s3')
>>> for bucket in s3.buckets.all():
...     print(bucket.name)
...
my-data-bucket-lv2
```

- This helps with the next commands to be able to use Python. Note that the command lines now changed to >>>
- For the previous error messages, I couldn’t have been able to reach the sodapy library because I did not have it downloaded in the amazon linux command line.

```
[ec2-user@ip-172-31-46-142 ~]$ pip3 install sodapy
```

```
>>> import boto3
>>> s3 = boto3.resource('s3')
>>> for bucket in s3.buckets.all():
...     print(bucket.name)
...
my-data-bucket-lv2
>>> import pandas as pd
>>> from sodapy import Socrata
>>> data_url='data.cityofnewyork.us'
>>> data_set='nc67-uf89'
>>> app_token='9niBdplRzQz7SpivsuzWjERln'
>>> client = Socrata(data_url,app_token)
>>> client.timeout = 60
>>> results = client.get(data_set)
>>> df = pd.DataFrame.from_records(results)
>>> df.to_csv('s3://my-data-bucket-lv2/NYC_Data.csv')
>>> quit()
```

[Link on how to use OpenData API]()

---

*Sent by the professor, on how to write the code for each year.

```
start = 0          # Start at 0
chunk_size = 2000     # Fetch 2000 rows at a time
results =[]          # Empty out our result list
where_clause="date_extract_y(created_date)=2017"
# See how many complaint records there are
record_count = client.get(data_set, where=where_clause, select="COUNT(*)")
# Loop until we have fetched all of the records
while True:
    # Fetch the set of records starting at 'start'
    results.extend( client.get(data_set, where=where_clause, offset=start, limit=chunk_size))
    # Move up the starting record
    start = start + chunk_size
    # If we have fetched all of the records, bail out
    if (start > int(record_count[0]['COUNT']) ):
      break
# Convert the list into a data frame
df = pd.DataFrame.from_records(results)
df.to_csv("annual_data_2017.csv",index=False)
```

Appendix A: Code for downloading or creating the data sets.

```
import boto3
s3=boto3.resource('s3')
for bucket in s3.buckets.all():
        print(bucket.name)

import sodapy
import pandas as pd
from sodapy import Socrata

data_url='data.cityofnewyork.us'
data_set='erm2-nwe9'
app_token='9niBdplRzQz7SpivsuzWjERln'
client = Socrata(data_url,app_token)
client.timeout = 60
start = 0
chunk_size = 2000
results =[]
where_clause="date_extract_y(created_date)=2017"
record_count = client.get(data_set, where=where_clause, select="COUNT(*)")
while True:
        results.extend( client.get(data_set, where=where_clause, offset=start, limit=chunk_size))
        start = start + chunk_size
        if (start > int(record_count[0]['COUNT'])):
                break

df = pd.DataFrame.from_records(results)
df.to_csv("annual_data_2017.csv",index=False)
```

Milestone 3:

In the EC2 Connect:

- I start by using the following code to be able to read the data.

```
aws s3 ls s3://my-data-bucket-lv2 --recursive --human-readable --summarize
```

- The next code is to look at the desired dataset.

```
aws s3 ls --summarize --human-readable --recursive
s3://my-data-bucket-lv2/Open_Parking_and_Camera_Violations.csv
```

- This code is to install it into python3

```
pip3 install boto3 pandas "s3fs<=0.4"
```

- The following code is the necessary pip installs needed and the libraries to be able to work with the data.

```
pip3 install boto3 pandas fsspec s3fs
python3
import boto3
```

import pandas as pd

- The following code is to read the dataset into pandas.

df = pd.read_csv('s3://my-data-bucket-lv2/Open_Parking_and_Camera_Violations.csv')

- I would then like to see the head of the dataset.

df.head()

```
>>> df.head()
   Unnamed: 0    plate state license_type  summons_number  ... county                   issuing_agency                                      summons_image  violation_status  judgment_entry_date
0           0  EVH1098    NY          PAS      4681370927  ...     QN  DEPARTMENT OF TRANSPORTATION  {'url': 'http://nycserv.nyc.gov/NYCServWeb/Sho...               NaN                  NaN
1           1  HJR9984    NY          PAS      4671631933  ...     QN  DEPARTMENT OF TRANSPORTATION  {'url': 'http://nycserv.nyc.gov/NYCServWeb/Sho...               NaN                  NaN
2           2  HTK5289    NY          PAS      4671632100  ...     QN  DEPARTMENT OF TRANSPORTATION  {'url': 'http://nycserv.nyc.gov/NYCServWeb/Sho...               NaN                  NaN
3           3  GGD3865    NY          PAS      4671632470  ...     BK  DEPARTMENT OF TRANSPORTATION  {'url': 'http://nycserv.nyc.gov/NYCServWeb/Sho...               NaN                  NaN
4           4   9JT379    MA          PAS      4671634843  ...     BK  DEPARTMENT OF TRANSPORTATION  {'url': 'http://nycserv.nyc.gov/NYCServWeb/Sho...               NaN                  NaN
```

- The next step I wanted to see was the different columns in the dataset.

df.columns

```
>>> df.columns
Index(['Unnamed: 0', 'plate', 'state', 'license_type', 'summons_number',
       'issue_date', 'violation_time', 'violation', 'fine_amount',
       'penalty_amount', 'interest_amount', 'reduction_amount',
       'payment_amount', 'amount_due', 'precinct', 'county', 'issuing_agency',
       'summons_image', 'violation_status', 'judgment_entry_date'],
      dtype='object')
```

- The next step: I would like more info into each columns

df.info

```
>>> df.info
<bound method DataFrame.info of      Unnamed: 0    plate state license_type  summons_number  ...             violation_status judgment_entry_date issue_year
0             0  EVH1098    NY          PAS      4681370927  ...                          NaN        NaN       2020          3    Tuesday
1             1  HJR9984    NY          PAS      4671631933  ...                          NaN        NaN       2019         11     Friday
2             2  HTK5289    NY          PAS      4671632100  ...                          NaN        NaN       2019         11     Friday
3             3  GGD3865    NY          PAS      4671632470  ...                          NaN        NaN       2019         11     Friday
4             4   9JT379    MA          PAS      4671634843  ...                          NaN        NaN       2019         11     Friday
..          ...      ...   ...          ...             ...  ...                          ...        ...        ...        ...        ...
995         995  HAU8944    NY          PAS      4658965914  ...                          NaN        NaN       2019          7     Monday
996         996  ZLK4988    PA          PAS      8617047744  ...     HEARING HELD-NOT GUILTY        NaN       2019          6    Tuesday
997         997    7L97B    NY          OMT      4665093878  ...                          NaN        NaN       2019          9   Thursday
998         998  CJR2160    NY          PAS      4665731342  ...                          NaN        NaN       2019          9  Wednesday
999         999  85777MC    NY          COM      8617047677  ...  HEARING HELD-GUILTY REDUCTION        NaN       2019          6     Monday
```

Appendix B Full source code for exploratory data analysis (descriptive statistics)

- .describe()

df.describe

```
>>> df.describe()
       Unnamed: 0  summons_number  fine_amount  penalty_amount  interest_amount  ...  payment_amount  amount_due    precinct
count  1000.000000    1.000000e+03   999.000000      999.000000       999.000000  ...      999.000000  999.000000  999.000000
mean    499.500000    5.237754e+09    55.200200        1.671672         0.100691  ...       53.340080    0.254254    6.269269
std     288.819436    1.527310e+09    16.724814        7.007605         2.520627  ...       18.075803    8.036204   20.071080
min       0.000000    2.003115e+09    35.000000        0.000000         0.000000  ...        0.000000    0.000000    0.000000
25%     249.750000    4.665713e+09    50.000000        0.000000         0.000000  ...       50.000000    0.000000    0.000000
50%     499.500000    4.671780e+09    50.000000        0.000000         0.000000  ...       50.000000    0.000000    0.000000
75%     749.250000    4.672028e+09    50.000000        0.000000         0.000000  ...       50.000000    0.000000    0.000000
max     999.000000    8.986658e+09   200.000000       60.000000        79.000000  ...      200.000000  254.000000  121.000000
```

- Next I wanted to take a closer look into the df columns named below: These are all financial keypoints.

df[["fine_amount","penalty_amount","interest_amount","reduction_amount","payment_amount", "amount_due"]].describe(include="all")

```
>>> df[["fine_amount","penalty_amount","interest_amount","reduction_amount","payment_amount","amount_due"]].describe(include="all")
       fine_amount  penalty_amount  interest_amount  reduction_amount  payment_amount  amount_due
count   999.000000      999.000000       999.000000        999.000000      999.000000  999.000000
mean     55.200200        1.671672         0.100691          3.378228       53.340080    0.254254
std      16.724814        7.007605         2.520627         14.475290       18.075803    8.036204
min      35.000000        0.000000         0.000000          0.000000        0.000000    0.000000
25%      50.000000        0.000000         0.000000          0.000000       50.000000    0.000000
50%      50.000000        0.000000         0.000000          0.000000       50.000000    0.000000
75%      50.000000        0.000000         0.000000          0.000000       50.000000    0.000000
max     200.000000       60.000000        79.000000        115.000000      200.000000  254.000000
```

- Next I wanted to see the data types of each of the columns

df.types

```
>>> df.dtypes
Unnamed: 0              int64
plate                  object
state                  object
license_type           object
summons_number          int64
issue_date             object
violation_time         object
violation              object
fine_amount           float64
penalty_amount        float64
interest_amount       float64
reduction_amount      float64
payment_amount        float64
amount_due            float64
precinct              float64
county                 object
issuing_agency         object
summons_image          object
violation_status       object
judgment_entry_date    object
```

- The best part of this dataset is that the data in the issue_date column is easy to use without any need of parsing.

print(df.issue_date)

```
>>> print(df.issue_date)
0      2020-03-17
1      2019-11-15
2      2019-11-15
3      2019-11-15
4      2019-11-15
          ...
995    2019-07-15
996    2019-06-18
997    2019-09-05
998    2019-09-11
999    2019-06-17
```

- This was to see the specific type of data for the issue date column

print(type(df.issue_date))

```
>>> print(type(df.issue_date))
<class 'pandas.core.series.Series'>
```

- Next is to use the datetime function to help me with the date

df.issue_date=pd.to_datetime(df.issue_date)
print(df.issue_date)

```
>>> df.issue_date=pd.to_datetime(df.issue_date)
>>> print(df.issue_date)
0     2020-03-17
1     2019-11-15
2     2019-11-15
3     2019-11-15
4     2019-11-15
         ...
995   2019-07-15
996   2019-06-18
997   2019-09-05
998   2019-09-11
999   2019-06-17
```

- I then checked to make sure that it is in the Datetime format

```
>>> print(type(df.issue_date[0]))
<class 'pandas._libs.tslibs.timestamps.Timestamp'>
```

print(type(df.issue_date[0]))

- I then took the year from each date and created a year column
- I then took the month from each date and created a month column
- I think took the day name of each date created a day_name column
- Finally I wanted it to describe

df['issue_year']=df.issue_date.dt.year
df['issue_month']=df.issue_date.dt.month
df['issue_day_name']=df.issue_date.dt.day_name()

df[['issue_year','issue_month','issue_day_name']].describe

```
>>> df['issue_year']=df.issue_date.dt.year
>>> df['issue_month']=df.issue_date.dt.month
>>> df['issue_day_name']=df.issue_date.dt.day_name()
```

```
>>> df[["issue_year","issue_month","issue_day_name"]].describe(include="all")
         issue_year  issue_month issue_day_name
count   1000.000000  1000.000000           1000
unique          NaN          NaN              7
top             NaN          NaN      Wednesday
freq            NaN          NaN            289
mean    2019.089000     9.177000            NaN
std        0.767899     2.053262            NaN
min     2000.000000     1.000000            NaN
25%     2019.000000     8.000000            NaN
50%     2019.000000     9.000000            NaN
75%     2019.000000    11.000000            NaN
max     2022.000000    12.000000            NaN
```

- Since I have created these new columns into df I would then like to have it saved onto my S3 bucket

>>>df.to_csv('s3://my-data-bucket-lv2/NYC_Data.csv', index=False)

>>>new_df = pd.read_csv('s3://my-data-bucket-lv2/NYC_Data.csv')

```
>>> new_df
     Unnamed: 0    plate state license_type  summons_number  ...             violation_status judgment_entry_date issue_year  issue_month issue_day_name
0             0  EVH1098    NY          PAS      4681370927  ...                          NaN                 NaN       2020            3        Tuesday
1             1  HJR9984    NY          PAS      4671631933  ...                          NaN                 NaN       2019           11         Friday
2             2  HTK5289    NY          PAS      4671632100  ...                          NaN                 NaN       2019           11         Friday
3             3  GGD3865    NY          PAS      4671632470  ...                          NaN                 NaN       2019           11         Friday
4             4   9JT379    MA          PAS      4671634843  ...                          NaN                 NaN       2019           11         Friday
..          ...      ...   ...          ...             ...  ...                          ...                 ...        ...          ...            ...
995         995  HAU8944    NY          PAS      4658965914  ...                          NaN                 NaN       2019            7         Monday
996         996  ZLK4988    PA          PAS      8617047744  ...      HEARING HELD-NOT GUILTY                 NaN       2019            6        Tuesday
997         997    7L97B    NY          OMT      4665093878  ...                          NaN                 NaN       2019            9       Thursday
998         998  CJR2160    NY          PAS      4665731342  ...                          NaN                 NaN       2019            9      Wednesday
999         999  85777MC    NY          COM      8617047677  ...  HEARING HELD-GUILTY REDUCTION                 NaN       2019            6         Monday
```

More statistical analysis on the data:

- Checking the value counts to see the different entries of data.
- Like expected NY would have the highest count of State license plate tickets for camera and parking violations

print(df.state.value_counts())

```
>>> print(df.state.value_counts())
NY    842
NJ     58
PA     24
IN     11
CT     10
FL      9
AZ      7
MD      6
GA      4
NC      4
CA      3
OH      3
MA      3
VA      3
RI      2
SC      2
TX      2
MO      1
NH      1
DC      1
IL      1
QB      1
TN      1
DP      1
```

- The next place to see the different arrays of license types, and of course as expected the most common type is Passenger type.

print(df.license_type.value_counts())

```
>>> print(df.license_type.value_counts())
PAS    805
COM     84
OMT     47
OMS     18
SRF     13
SCL      8
TRC      5
ORG      4
APP      3
PSD      3
TOW      2
OMR      2
LMB      1
SPO      1
MED      1
MOT      1
RGL      1
HIS      1
```

- I then wanted to see the different days that had more value counts. Surprisingly I would have imagined a weekend day to be more frequent like (Friday, Saturday or Sunday) but Wednesday was the highest!

print(df.issue_day_name.value_counts())

```
>>> print(df.issue_day_name.value_counts())
Wednesday    289
Friday       235
Monday       156
Tuesday      131
Thursday     119
Saturday      49
Sunday        21
```

- Next were just value counts, just to see the trends in the data.

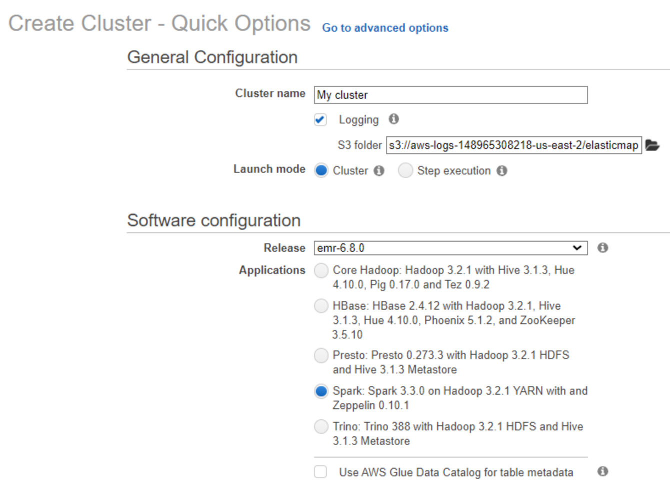
print(df.issue_year.value_counts())
print(df.issue_month.value_counts())

Milestone 4:

Coding and Modeling: Write the PySpark code to read and process this data using an AWS EMR cluster. This will include code to read the source data, clean and normalize the data, feature engineering, training/testing and evaluation of the predictive models and output. Results of the analysis should be written to a file (or a series of files). Include all source code and proper references to each of the libraries/modules you are using. The resulting code should be able to be automated (scripted). Complete this section with a brief paragraph summarizing the main steps your program takes and any challenges you may have encountered while cleaning and processing the data.

In the EMR connecting with the instance:

Creating a cluster:



- Once creating the cluster, we want to connect to the master node.

Pyspark

# Import some functions we will need later on from pyspark.sql.functions import col, isnan, when, count, udf # Set the Spark logging level to only show errors sc.setLogLevel("ERROR")
bucket = 'my-data-bucket-lv2'
filename = 'Open_Parking_and_Camera_Violations.csv'
file_path = 's3a://' + bucket + filename
sdf = spark.read.csv(file_path, sep=',', header=True, inferSchema=True)
sdf.printSchema()

```
[hadoop@ip-172-31-44-181 ~]$ pyspark
Python 3.7.10 (default, Jun  3 2021, 00:02:01)
[GCC 7.3.1 20180712 (Red Hat 7.3.1-13)] on linux
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
22/11/15 04:28:15 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is set, falling back to uploading libraries under SPARK_HOME.
Welcome to
      ____              __
     / __/__  ___ _____/ /__
    _\ \/ _ \/ _ `/ __/  '_/
   /__ / .__/\_,_/_/ /_/\_\   version 3.3.0-amzn-0
      /_/

Using Python version 3.7.10 (default, Jun  3 2021 00:02:01)
Spark context Web UI available at http://ip-172-31-44-181.us-east-2.compute.internal:4040
Spark context available as 'sc' (master = yarn, app id = application_1668480946024_0003).
SparkSession available as 'spark'.
>>> from pyspark import SparkContext, SparkConf
>>>
>>> from pyspark.sql.functions import col, isnan, when, count, udf
>>>
>>> sc.setLogLevel("ERROR")
>>>
>>> bucket = 'my-data-bucket-lv2'
>>> filename = 'Open_Parking_and_Camera_Violations.csv'
>>> file_path = 's3a://' + bucket + filename
>>>
>>> sdf = spark.read.csv(file_path, sep=',', header=True, inferSchema=True)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/spark/python/pyspark/sql/readwriter.py", line 535, in csv
    return self._df(self._jreader.csv(self._spark._sc._jvm.PythonUtils.toSeq(path)))
  File "/usr/lib/spark/python/lib/py4j-0.10.9.5-src.zip/py4j/java_gateway.py", line 1322, in __call__
  File "/usr/lib/spark/python/pyspark/sql/utils.py", line 190, in deco
    return f(*a, **kw)
  File "/usr/lib/spark/python/lib/py4j-0.10.9.5-src.zip/py4j/protocol.py", line 328, in get_return_value
```

- To begin I start a New Cluster
- Add the Cluster Inbound Rules to allow the SSH client to work anywhere.
- Then I restart my instances
- Connect to the EMR Cluster Master Instance to be able to work within python and pyspark.

pyspark
import pyspark
sc.setLogLevel("ERROR")
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, isnan, when, count, udf
df=spark.read.csv('s3://my-data-bucket-lv2/Open_Parking_and_Camera_Violations.csv', header=True, inferSchema= True)
df.show(5)

```
|  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|       Summons Image|
|KDS2339|   NY|         PAS|    4718325341|12/24/2020|        06:55P|PHTO SCHOOL ZN SP...|               null|       50.0|           0.0|            0.0|             0.0|          50.0|       0.0|       0|    MN|DEPARTMENT OF TRA...|
           null|View Summons (htt...|
|HPC9489|   NY|         PAS|    4696317389|06/22/2020|        04:52P|PHTO SCHOOL ZN SP...|               null|       50.0|          25.0|            0.0|             0.0|          75.0|       0.0|       0|    QN|DEPARTMENT OF TRA...|
           null|View Summons (htt...|
|JKL9363|   NY|         PAS|    1480785740|12/28/2020|        07:50A|NO PARKING-STREET...|               null|       65.0|           0.0|            0.0|             0.0|          65.0|       0.0|      66|     K|DEPARTMENT OF SAN...|HEA
RING HELD-GUILTY|View Summons (htt...|
|JKF2691|   NY|         PAS|    5117181335|09/17/2021|        02:53P|FAILURE TO STOP A...|               null|       50.0|           0.0|            0.0|             0.0|          50.0|       0.0|       0|    QN|DEPARTMENT OF TRA...|
           null|View Summons (htt...|
|JPM6052|   NY|         PAS|    5117181384|09/17/2021|        02:59P|FAILURE TO STOP A...|               null|       50.0|           0.0|            0.0|             0.0|          50.0|       0.0|       0|    BX|DEPARTMENT OF TRA...|
           null|View Summons (htt...|
```

- Working in the EMR Cluster in Python **
- To install the necessary pip installs to be able to make the visualizations needed.

pip3 install boto3 pandas "s3fs<=0.4"
pip3 install boto3 pandas fsspec s3fs

python3 -m pip install matplotlib
python3 -m pip install seaborn

pyspark
sc.setLogLevel("ERROR")
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, isnan, when, count, udf
from pyspark.ml.stat import Correlation
from pyspark.sql.functions import *

import boto3
import pandas as pd
import io
import s3fs

- This then worked under PySpark **

df=spark.read.csv('s3://my-data-bucket-lv2/Open_Parking_and_Camera_Violations.csv', header=True, inferSchema= True)

df.show(5)
df.printSchema()

```
|  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|       Summons Image|
|KDS2339|   NY|         PAS|    4718325341|12/24/2020|        06:55P|PHTO SCHOOL ZN SP...|               null|       50.0|           0.0|            0.0|             0.0|          50.0|       0.0|       0|    MN|DEPARTMENT OF TRA...|
           null|View Summons (htt...|
|HPC9489|   NY|         PAS|    4696317389|06/22/2020|        04:52P|PHTO SCHOOL ZN SP...|               null|       50.0|          25.0|            0.0|             0.0|          75.0|       0.0|       0|    QN|DEPARTMENT OF TRA...|
           null|View Summons (htt...|
|JKL9363|   NY|         PAS|    1480785740|12/28/2020|        07:50A|NO PARKING-STREET...|               null|       65.0|           0.0|            0.0|             0.0|          65.0|       0.0|      66|     K|DEPARTMENT OF SAN...|HEA
RING HELD-GUILTY|View Summons (htt...|
|JKF2691|   NY|         PAS|    5117181335|09/17/2021|        02:53P|FAILURE TO STOP A...|               null|       50.0|           0.0|            0.0|             0.0|          50.0|       0.0|       0|    QN|DEPARTMENT OF TRA...|
           null|View Summons (htt...|
|JPM6052|   NY|         PAS|    5117181384|09/17/2021|        02:59P|FAILURE TO STOP A...|               null|       50.0|           0.0|            0.0|             0.0|          50.0|       0.0|       0|    BX|DEPARTMENT OF TRA...|
           null|View Summons (htt...|
```

- I then tried to start parsing the Issue Date, I wanted to create a column for the day of the week, year and months. **
- I ran the following code, but it never loaded. **

Appendix C Full source code for the ML pipeline (cleaning, feature extraction, model building, etc.)

aws s3 cp
s3://nypd-open-parking-camera-violations/Open_Parking_and_Camera_Violations_2022.cs
v.gz .
aws s3 cp
s3://nypd-open-parking-camera-violations/Open_Parking_and_Camera_Violations_2021.cs
v.gz .
aws s3 cp
s3://nypd-open-parking-camera-violations/Open_Parking_and_Camera_Violations_2020.cs
v.gz .

- I did this for each year.

```
[hadoop@ip-172-31-30-53 ~]$ pwd
/home/hadoop
[hadoop@ip-172-31-30-53 ~]$ ls -l
total 1960440
-rw-rw-r-- 1 hadoop hadoop 238269337 Dec 10 22:52 Open_Parking_and_Camera_Violations_2017.csv.gz
-rw-rw-r-- 1 hadoop hadoop 246404905 Dec 10 22:52 Open_Parking_and_Camera_Violations_2018.csv.gz
-rw-rw-r-- 1 hadoop hadoop 369406870 Dec 13 23:57 Open_Parking_and_Camera_Violations_2019.csv.gz
-rw-rw-r-- 1 hadoop hadoop 352933013 Dec 13 23:57 Open_Parking_and_Camera_Violations_2020.csv.gz
-rw-rw-r-- 1 hadoop hadoop 405528259 Dec 12 21:58 Open_Parking_and_Camera_Violations_2021.csv.gz
-rw-rw-r-- 1 hadoop hadoop 394934626 Dec 12 21:58 Open_Parking_and_Camera_Violations_2022.csv.gz
```

```
hdfs dfs -put  Open_Parking_and_Camera_Violations_2022.csv.gz   hdfs:///
hdfs dfs -put  Open_Parking_and_Camera_Violations_2021.csv.gz   hdfs:///
hdfs dfs -put  Open_Parking_and_Camera_Violations_2020.csv.gz   hdfs:///
hdfs dfs -put  Open_Parking_and_Camera_Violations_2019.csv.gz   hdfs:///
hdfs dfs -put  Open_Parking_and_Camera_Violations_2018.csv.gz   hdfs:///
hdfs dfs -put  Open_Parking_and_Camera_Violations_2017.csv.gz   hdfs:///
```

```
Pyspark
>>> import pyspark
>>> sc.setLogLevel("ERROR")
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.functions import col, isnan, when, count, udf
>>>sdf = spark.read.csv("hdfs:///Open_Parking_and_Camera_Violations_2022.csv.gz",
header=True, inferSchema=True)
>>> sdf.show()
>>>sdf.select(sdf.Issue_Date, sdf.Violation_Time, sdf.Judgment_Entry_Date).show()
```

```
+----------+--------------+-------------------+
|Issue_Date|Violation_Time|Judgment_Entry_Date|
+----------+--------------+-------------------+
|03/07/2022|        11:25A|         07/28/2022|
|02/03/2022|        01:43P|         07/28/2022|
|04/12/2022|        05:21A|         07/28/2022|
|04/12/2022|        08:51A|         07/28/2022|
|04/25/2022|        11:21A|         08/11/2022|
|04/26/2022|        05:14A|               null|
|03/23/2022|        12:23P|               null|
|04/27/2022|        10:56A|         08/11/2022|
|03/30/2022|        01:42P|         07/28/2022|
|04/01/2022|        05:00P|         07/28/2022|
|05/28/2022|        06:40A|         09/15/2022|
|01/06/2022|        04:07P|         07/28/2022|
|03/16/2022|        03:26P|         09/01/2022|
|05/27/2022|        10:50A|         09/15/2022|
|06/30/2022|        08:59A|         10/20/2022|
|04/23/2022|        06:55A|         10/06/2022|
|04/29/2022|        11:11A|               null|
|07/27/2022|        08:31A|         11/10/2022|
|07/30/2022|        09:08A|               null|
|08/09/2022|        08:39A|               null|
+----------+--------------+-------------------+
```

>>> sdf.printSchema()

```
>>> sdf.printSchema()
root
 |-- Plate: string (nullable = true)
 |-- State: string (nullable = true)
 |-- License_Type: string (nullable = true)
 |-- Summons_Number: long (nullable = true)
 |-- Issue_Date: string (nullable = true)
 |-- Violation_Time: string (nullable = true)
 |-- Violation: string (nullable = true)
 |-- Judgment_Entry_Date: string (nullable = true)
 |-- Fine_Amount: double (nullable = true)
 |-- Penalty_Amount: double (nullable = true)
 |-- Interest_Amount: double (nullable = true)
 |-- Reduction_Amount: double (nullable = true)
 |-- Payment_Amount: double (nullable = true)
 |-- Amount_Due: double (nullable = true)
 |-- Precinct: integer (nullable = true)
 |-- County: string (nullable = true)
 |-- Issuing_Agency: string (nullable = true)
 |-- Violation_Status: string (nullable = true)
 |-- Summons_Image: string (nullable = true)
```

This helps with the date, and time. This allows us to see the dates and time periods more coherently.

>>> from pyspark.sql.functions import *
>>> sdf = sdf.withColumn("Issue_Date", to_date(col("Issue_Date"), "MM/dd/yyyy"))
>>> sdf = sdf.withColumn("Judgment_Entry_Date", to_date(col("Judgment_Entry_Date"), "MM/dd/yyyy"))
>>> sdf = sdf.withColumn("Violation_Time", regexp_replace("Violation_Time", '\.', '0'))
>>> sdf = sdf.withColumn("Violation_Time", regexp_replace("Violation_Time", ' ', '0'))
>>> sdf = sdf.withColumn("Violation_Time", when(col("Violation_Time").rlike("A|P"), col("Violation_Time")).otherwise(concat(col("Violation_Time"), lit("A"))))

```
>>> sdf = sdf.withColumn("date_and_time", concat( date_format(col("Issue_Date"), "yyyy-MM-dd"), lit(" "), col("Violation_Time"), lit("M")) )
>>> sdf = sdf.withColumn("Issue_Date_Time", to_timestamp(col("date_and_time"), "yyyy-MM-dd hh:mma"))
>>> sdf = sdf.drop("date_and_time")
>>> sdf.show()
```

>>> sdf.show()

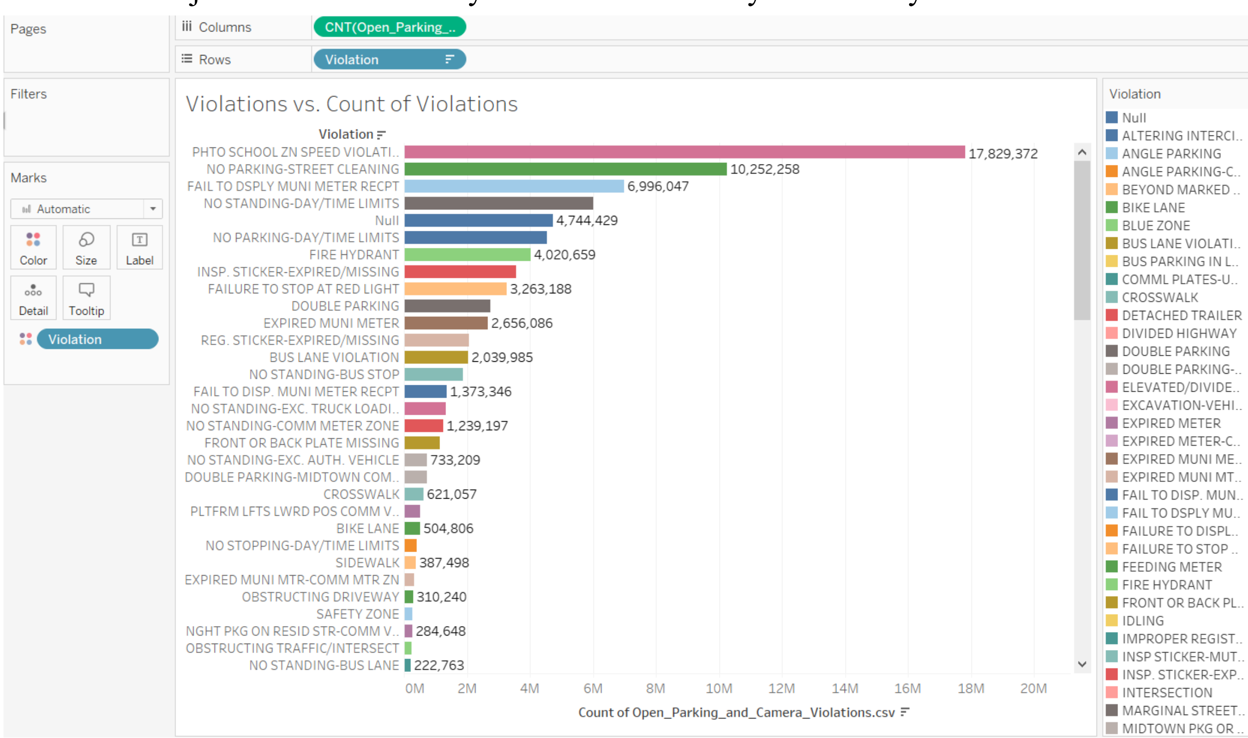
Plate	State	License_Type	Summons_Number	Issue_Date	Violation_Time	Violation	Judgment_Entry_Date	Fine_Amount	Penalty_Amount	Interest_Amount	Reduction_Amount	Payment_Amount	Amount_Due	Precinct	County	Issuing_Agency	Violation_Status	Summons_Image	Issue_Date_Time
HKP9056	NY	PAS	8690435980	2022-03-07	11:25A	INSP. STICKER-EXP...	2022-07-28	65.0	60.0	3.2	0.06	128.14	0.0	70	K	TRAFFIC	null	View Summons (htt...	2022-03-07 11:25:00
XGVN10	NJ	PAS	8702035650	2022-02-03	01:43P	NO PARKING-DAY/TI...	2022-07-28	65.0	60.0	3.2	0.0	128.2	0.0	20	NY	TRAFFIC	null	View Summons (htt...	2022-02-03 13:43:00
T696628C	NY	OMT	8690445201	2022-04-12	05:21A	SIDEWALK	2022-07-28	115.0	60.0	5.19	0.0	0.0	180.19	69	K	TRAFFIC	null	View Summons (htt...	2022-04-12 05:21:00
KDJ1693	NY	PAS	8690445470	2022-04-12	08:51A	NO PARKING-STREET...	2022-07-28	65.0	60.0	2.4	2.25	125.15	0.0	69	K	TRAFFIC	null	View Summons (htt...	2022-04-12 08:51:00
LNE769	OK	PAS	8690447945	2022-04-25	11:21A	DOUBLE PARKING	2022-08-11	115.0	60.0	2.71	0.21	177.5	0.0	78	K	TRAFFIC	HEARING HELD-GUILTY	View Summons (htt...	2022-04-25 11:21:00
KRK7051	NY	PAS	8690448196	2022-04-26	05:14A	FIRE HYDRANT	null	115.0	10.0	0.0	0.0	125.0	0.0	75	K	TRAFFIC	null	View Summons (htt...	2022-04-26 05:14:00
35264NC	NY	COM	8702045280	2022-03-23	12:23P	COMML PLATES-UNAL...	null	115.0	10.0	0.0	0.0	125.0	0.0	19	NY	TRAFFIC	HEARING HELD-GUIL...	View Summons (htt...	2022-03-23 12:23:00
KTX9233	NY	PAS	8702335580	2022-04-27	10:56A	NO STOPPING-DAY/T...	2022-08-11	115.0	60.0	3.86	0.17	178.69	0.0	23	NY	TRAFFIC	null	View Summons (htt...	2022-04-27 10:56:00
KJW1904	NY	PAS	8702046854	2022-03-30	01:42P	FIRE HYDRANT	2022-07-28	115.0	60.0	5.06	0.17	179.89	0.0	19	NY	TRAFFIC	null	View Summons (htt...	2022-03-30 13:42:00
KSV9601	NY	PAS	8702047469	2022-04-01	05:00P	FRONT OR BACK PLA...	2022-07-28	65.0	60.0	3.58	0.22	128.36	0.0	23	NY	TRAFFIC	null	View Summons (htt...	2022-04-01 17:00:00
KTX9233	NY	PAS	8702342870	2022-05-28	06:40A	NO STANDING-BUS STOP	2022-09-15	115.0	60.0	2.36	0.17	177.19	0.0	23	NY	TRAFFIC	null	View Summons (htt...	2022-05-28 06:40:00
CII6082	GA	PAS	8690898890	2022-01-06	04:07P	EXPIRED MUNI METER	2022-07-28	35.0	60.0	1.74	0.09	96.65	0.0	108	Q	TRAFFIC	HEARING HELD-GUILTY	View Summons (htt...	2022-01-06 16:07:00
P37HDD	NJ	PAS	8702327314	2022-03-16	03:26P	FIRE HYDRANT	2022-09-01	115.0	60.0	3.41	0.3	178.11	0.0	20	NY	TRAFFIC	HEARING HELD-GUILTY	View Summons (htt...	2022-03-16 15:26:00
AW83009	CT	PAS	8769247500	2022-05-27	10:50A	NO STANDING-EXC. ...	2022-09-15	95.0	60.0	1.62	0.19	156.43	0.0	112	Q	TRAFFIC	null	View Summons (htt...	2022-05-27 10:50:00
HTK7380	NY	PAS	8769248515	2022-06-30	08:59A	NO PARKING-STREET...	2022-10-20	65.0	60.0	0.21	0.15	125.06	0.0	103	Q	TRAFFIC	null	View Summons (htt...	2022-06-30 08:59:00
JMR9201	NY	PAS	8702334781	2022-04-23	06:55A	REG. STICKER-EXPI...	2022-10-06	65.0	60.0	1.44	0.0	126.44	0.0	28	NY	TRAFFIC	HEARING HELD-GUILTY	View Summons (htt...	2022-04-23 06:55:00

```
>>> sdf.agg(min(col("Fine_Amount"))).show()
+----------------+
|min(Fine_Amount)|
+----------------+
|             0.0|
+----------------+
>>> sdf.agg(max(col("Fine_Amount"))).show()
+----------------+
|max(Fine_Amount)|
+----------------+
|           515.0|
+----------------+
>>> sdf.agg(min(col("Amount_Due"))).show()
+---------------+
|min(Amount_Due)|
+---------------+
|            0.0|
+---------------+
>>> sdf.agg(max(col("Amount_Due"))).show()
+---------------+
|max(Amount_Due)|
+---------------+
|         592.13|
+---------------+
```

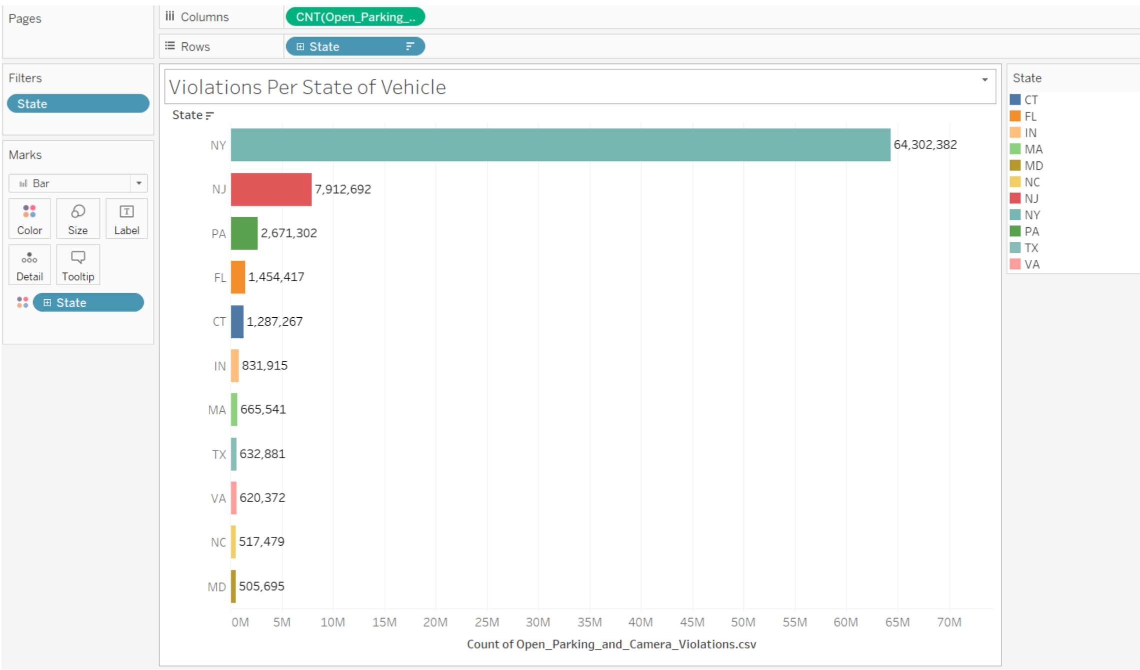
Appendix D Full source code for Visualization

Milestone 5:

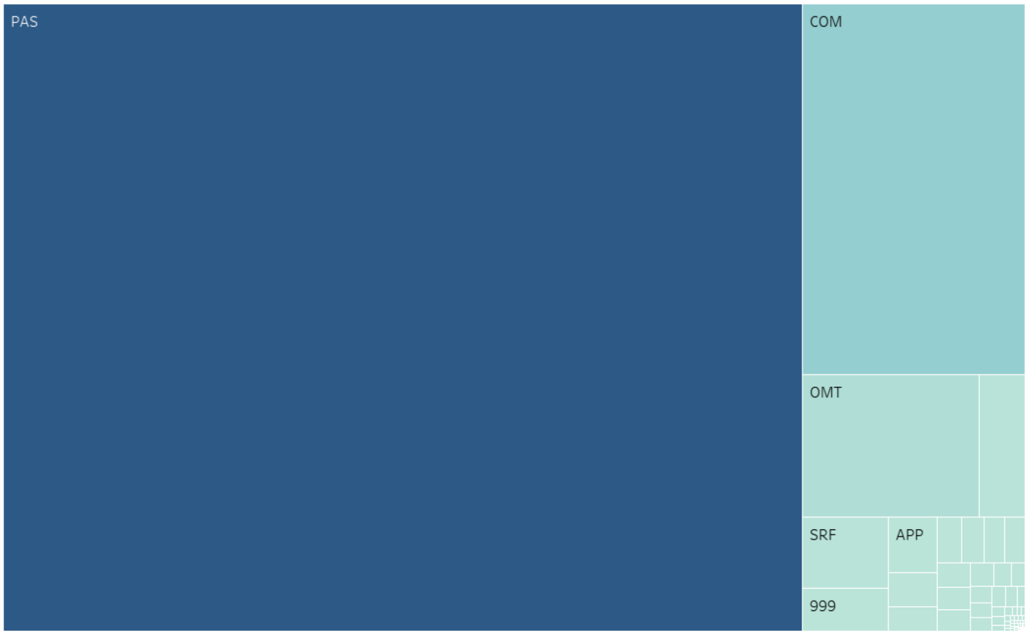
- **The first graph:**
    - I wanted to see the various traffic violations. I wanted to also see the traffic violations that were more common than others.
    - The most frequent violations are **Photo School Zone Speed Violations**. Over 17M accounts for this violation.
    - The next most frequent violation is the **No Parking Street Cleaning.** Over 10M accounts for this violation.
    - The third most frequent violation is the **Fail to Display Municipal Meter Receipt**. Over 6M accounts for this violation.
    - Given this data, and visualization, we can see that when it comes to the cameras in NYC, the Photo School Zone Speed Violations are the ones that are strictly enforced in NYC. NYC has been known to be a fast paced city, and cars drive above the speed limit almost all the time, so reinforcing these violations are necessary especially in school zones.
    - The No Parking due to street cleaning is another good violation to have, since NYC has had so many issues with cleaning, rats etc. Recently this has been one of the major problems with NYC due to infestation of rodents and unwanted creatures in households, buildings, restaurants.
    - Most of these violations are repeated throughout the same violations. They are just worded differently or entered differently into their system.



- **The second graph:**
    - I wanted to see the various states that the drivers accounted for in violations.
    - To my surprise, Florida had more violations than Connecticut. The tri-state area had the most like: NY, NJ, PA. This is all information based on the plate of the car. So if a car had Florida plates, we don't know the driver's information.
    - It could have been a civilian, or it could be out of state visitors.

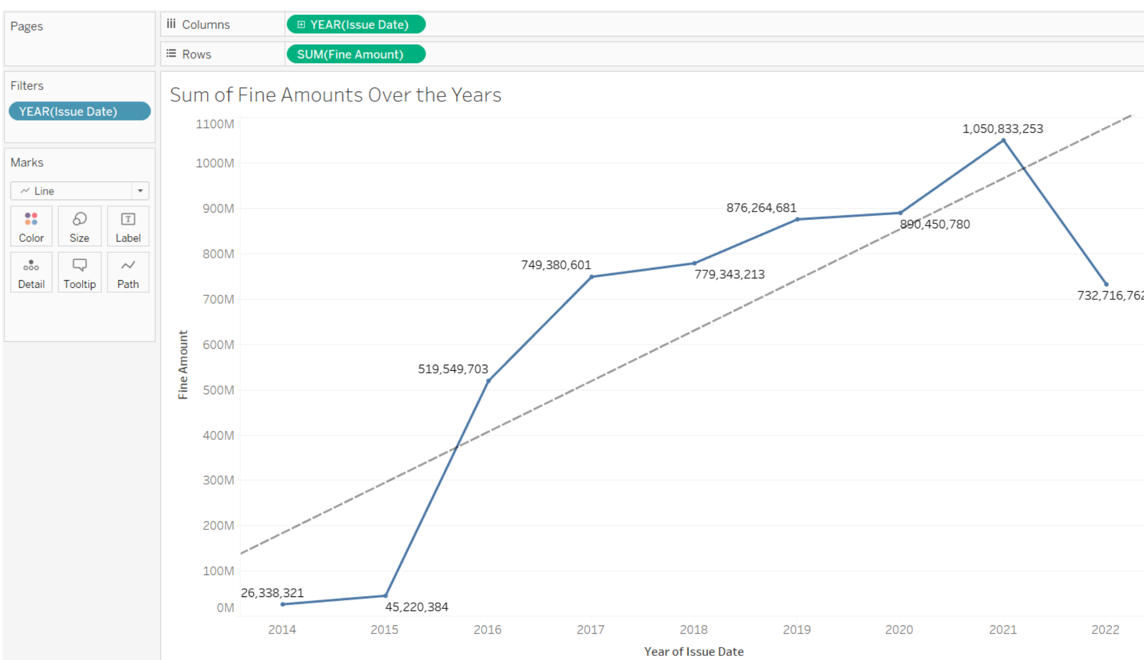


- **The third graph:**
    - I just wanted to see the different license types Passenger and Commercials were the most common among the rest of the license types.



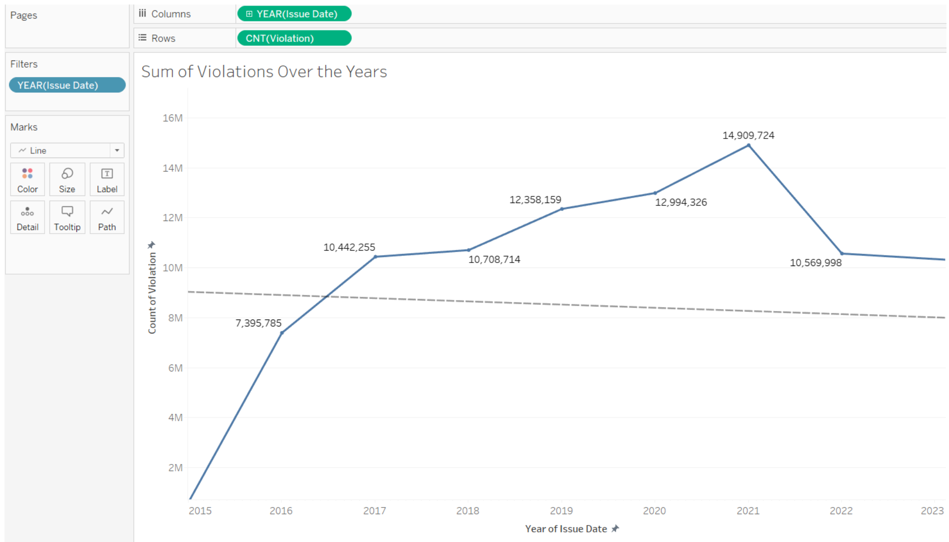
The Fourth Graph:

- I then just wanted to see the sum amounts of fines over the years. The year of 2021 there was over 1M of fine amounts for the year 2021. There has been a drop in 2022 for the fine amounts. It has significantly decreased by about ~300K.
- The trendline accounts for the increase in fine amounts over the years. Within the year there has been a drop where we can see that they have either given less violations or fine amounts have decreased.



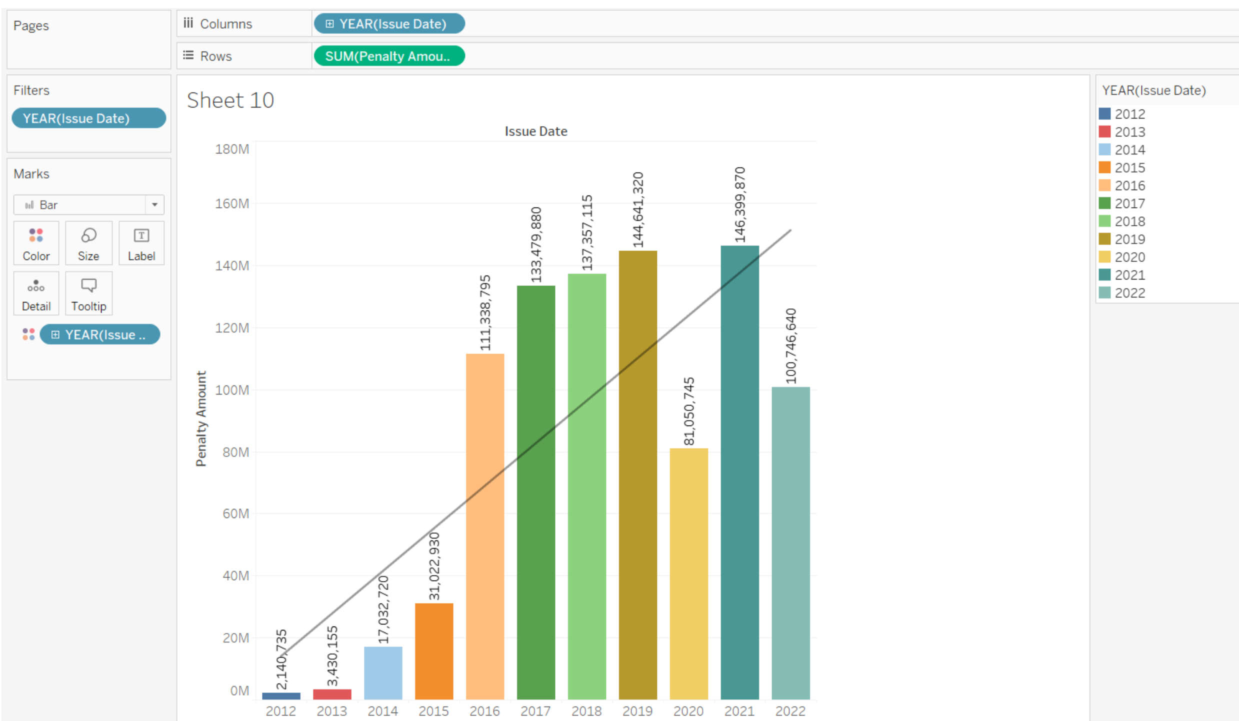
- When describing the last graph: I then wanted to see the violations per year and see if there was a correlation with the fine amounts of the previous graph.

- There has been a significant decrease in the amount of violations per year. 2021 being the year that they had the most.

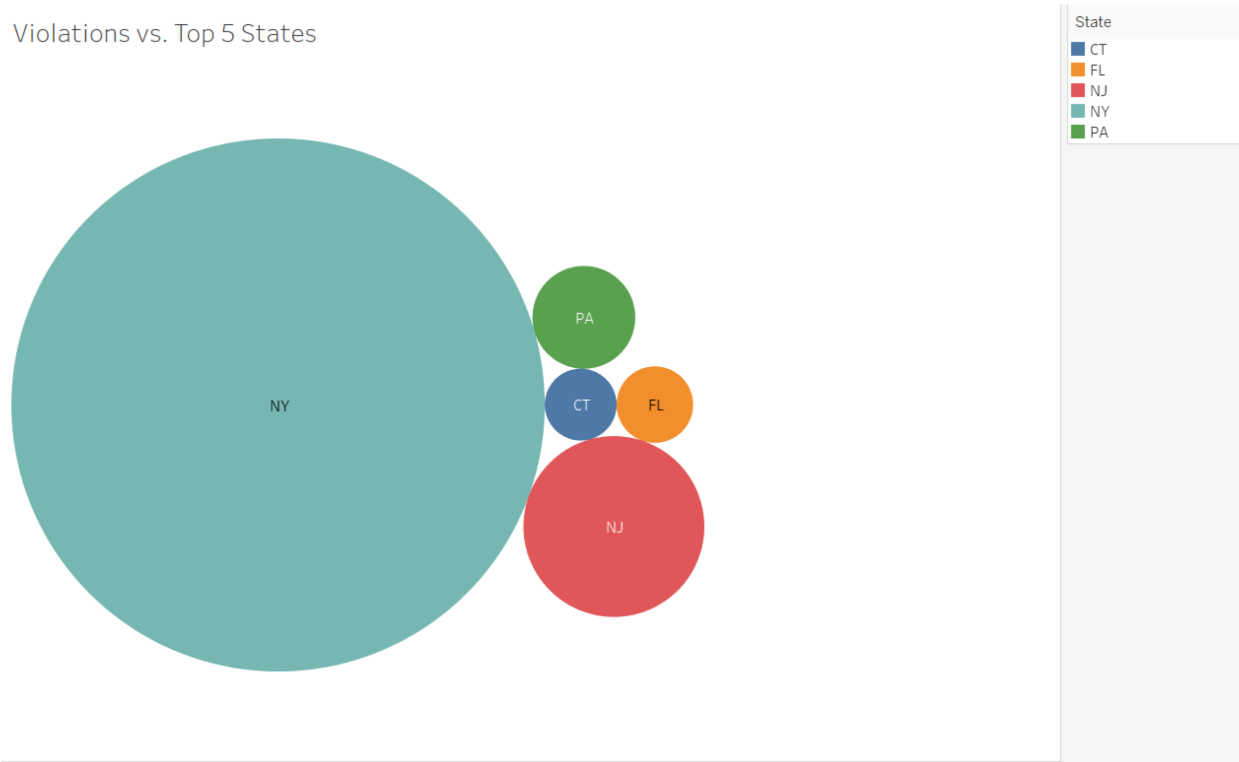


-

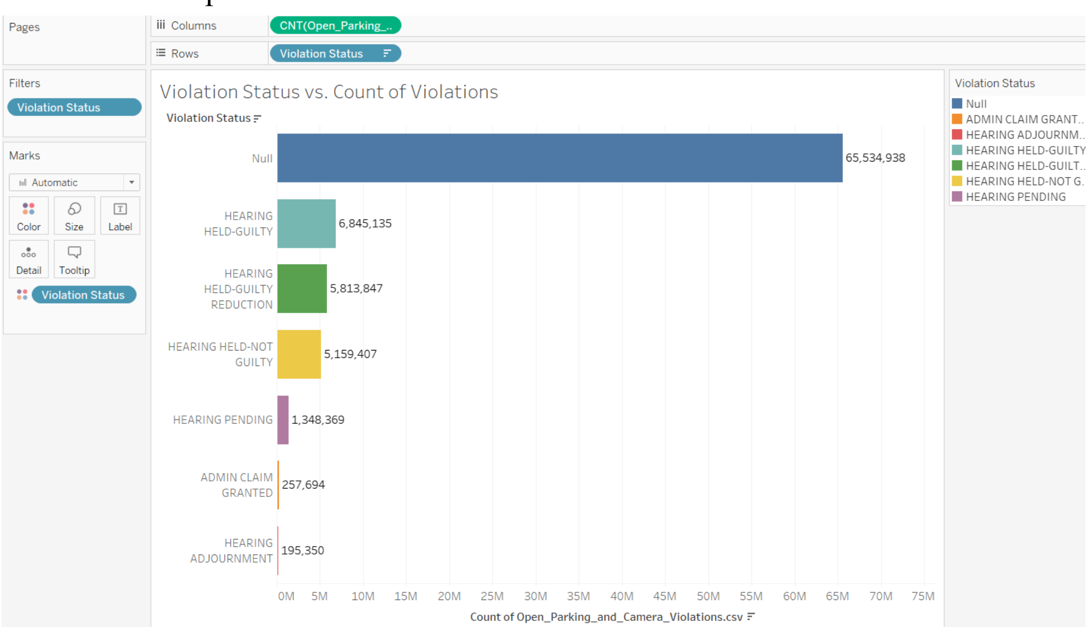
- Another graph to further look into the penalty amounts. During 2021, there were a significant number of violations. As the years have gone by so have the penalty sums.



- The 5th Graph:
    - This is a graph that better depicts graph 2:

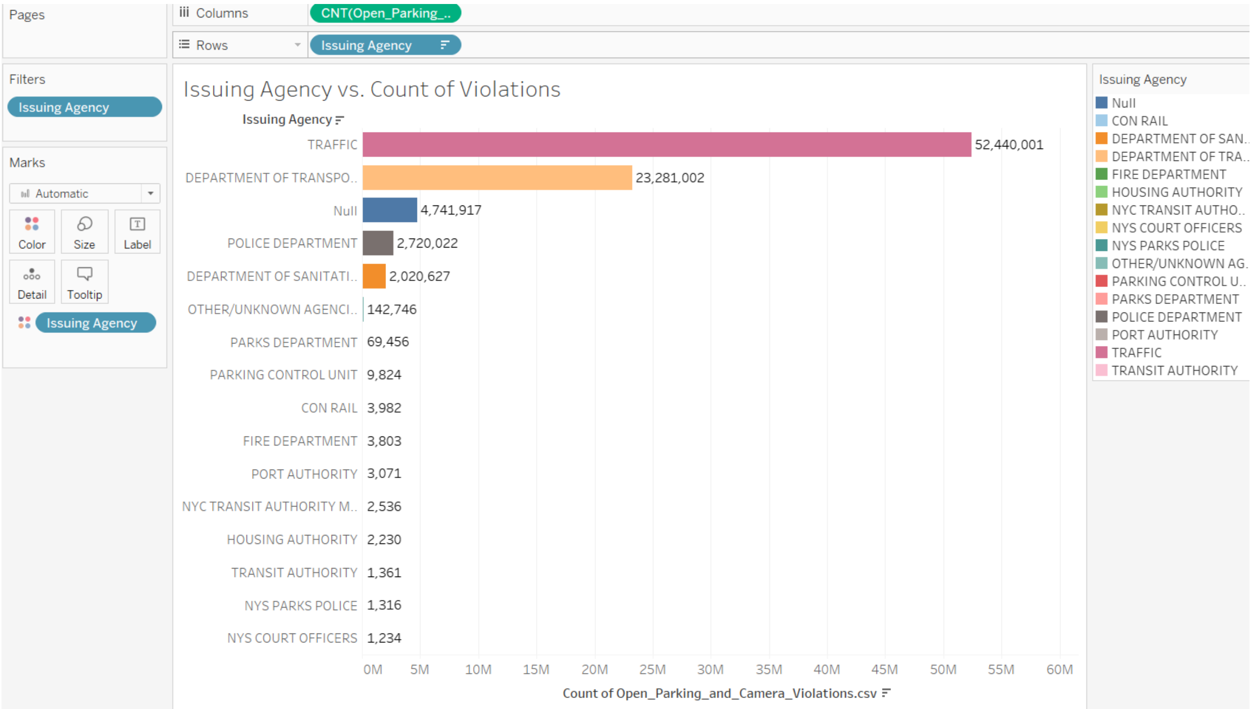


- The 6th Graph:
    - The sixth graph shows the different violation statuses and their updates. The main issue with their system is that it has null values. Most of the violations are not updated.



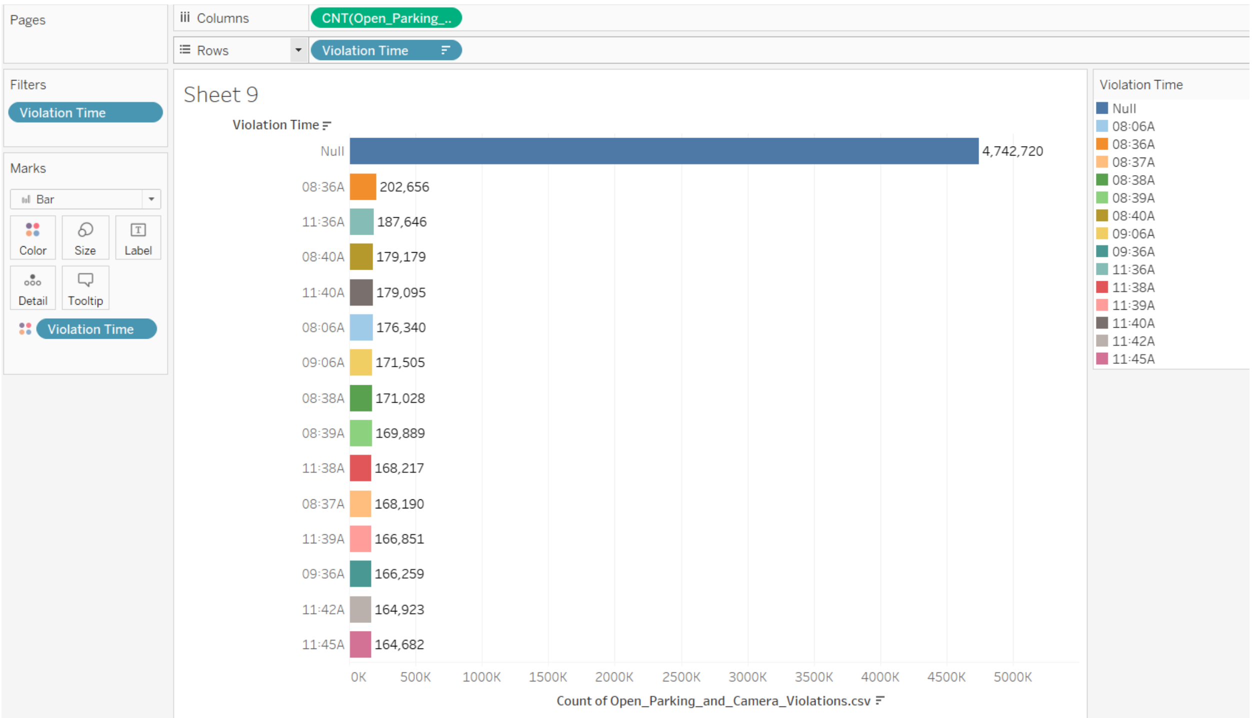
7th Graph:

- The agency who issues the violations for the most part is the Traffic Agency. This would account for the most of the violations.



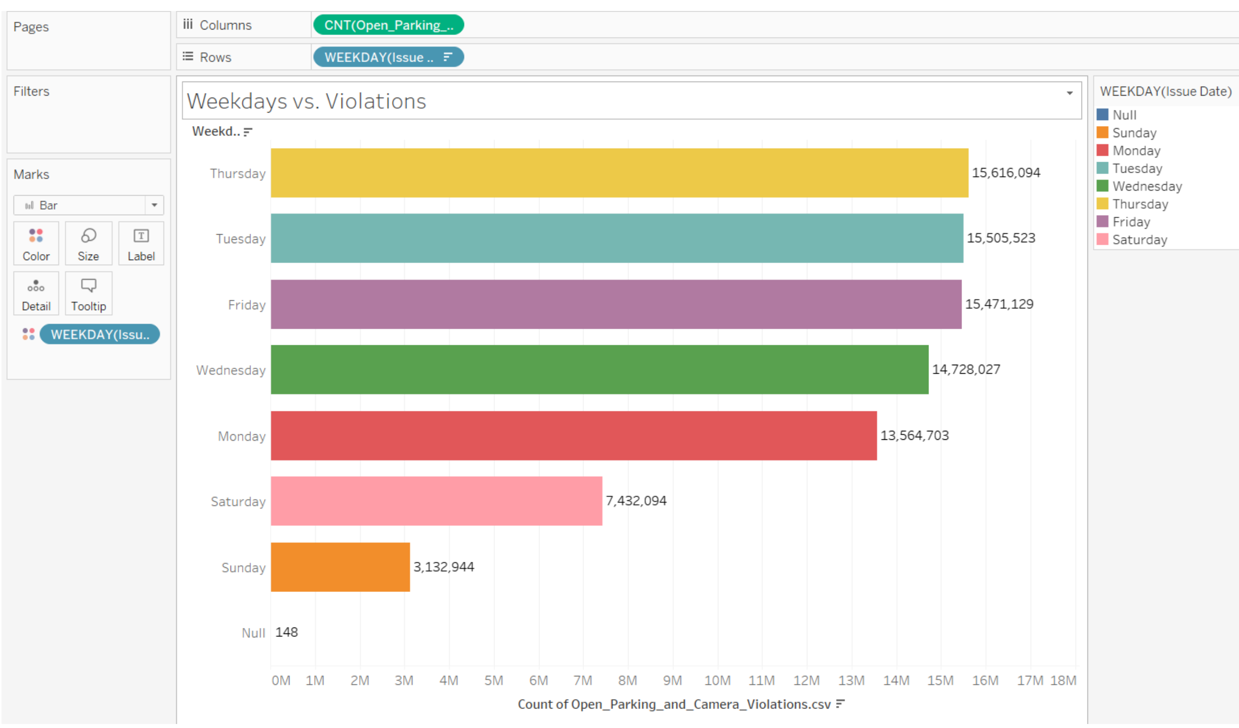
8th Graph:

- I wanted to see a trend for the violation times.
- Unfortunately there is a large sum of data that does not have any time stamps.
- For the most part, we can see that the most frequent hours of traffic violations are during 8:06AM, 8:36AM and 8:37AM, 8:38AM, and 8:39AM. There are some 9AM ones and 11AM getting closer to noon of lunch hour.
- The early 8AM ones could be traveling to work, school so it does make sense if someone could be rushing to get to their destination.



9th Graph:

- I wanted to see which day of the week most violations occur. To my surprise it was Thursday, Tuesday and Friday in third.
- From my initial theories, I believed that week-ends would be the most frequent violations.



Milestone 6 & 7:

Summary and conclusions:

Retrieving the data came with its own complications. Since my data was on the NYC Open Data. To be able to get access to the data, I needed to create an OpenData account to be able to retrieve the data from the API. Once having access to the API I was then able to download the data by the year. Finally the downloaded data is then in my S3 bucket to be able to open the CSV files.
From there I was able to use AWS to process the data file. I was then able to look at the data set, manipulate the dataset for it to work to my standards. Being able to manipulate the data from the times and the dates help the overall dataset to come up with conclusions on the data. The visualizations that I have created were created via Tableau. Tableau was very helpful in the making of the visualizations. From the data, initially it was believed that there were more violations made during the weekend days, but from the visualizations we could see that the most frequent day was on Thursday, then on Tuesday. The different arrays of timeframes in the mornings that were saturated with violations were during the morning times. This could account for rush hours of trying to get to school and/or work.