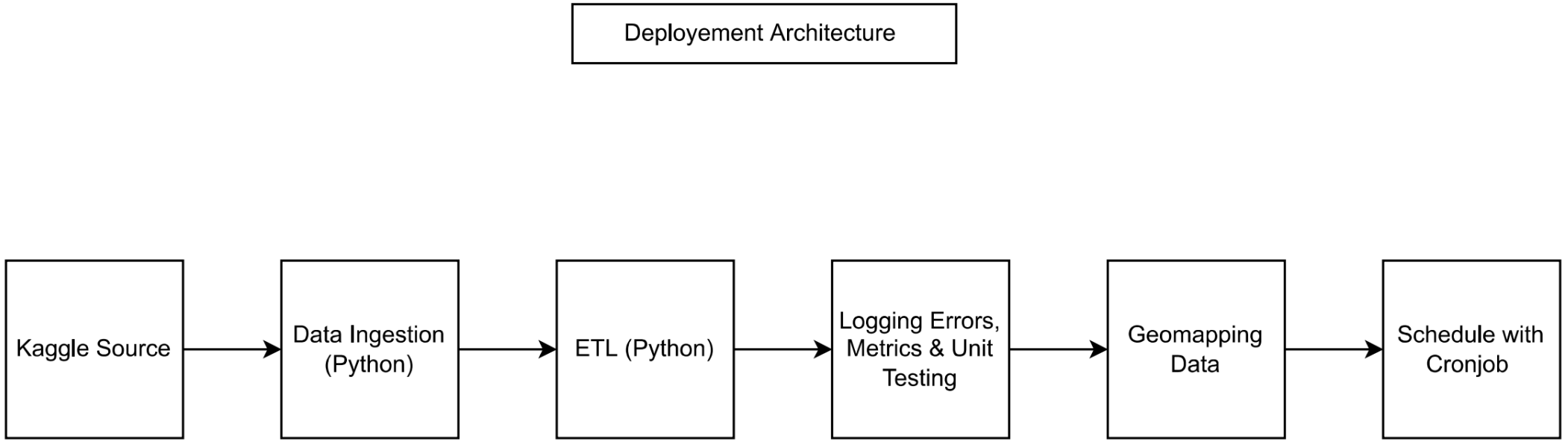
Geomapping of Traffic Violations

The Overview

The purpose of this project is to create a visual representation of the data using geomapping. These mappings can offer a better understanding of the data and be potentially utilized for research purposes.

The Pipeline



The Data Source

Source of the data:

https://www.kaggle.com/code/microtang/exploration-the-violations/data

Kaggle

- In order to utlizie data from Kaggle we must first install and authenticate Kaggle’s public API.

- The instructions can be found here:
- https://www.kaggle.com/docs/api

The Extraction

- Data is downloaded from the websource and stored locally.
- Data is then extracted from the zip file.

Download data from source and store locally. The file is then inzipped so the data can be viewed.

In [36]:

```
!kaggle datasets download --force -d felix4guti/traffic-violations-in-usa
print('downloaded data')
with zipfile.ZipFile("./{}.zip".format("traffic-violations-in-usa"),"r") as z:
    z.extractall(".")
print('unzipped the data')
```

```
  0%|          | 0.00/48.5M [00:00<?, ?B/s]
  2%|2         | 1.00M/48.5M [00:00<00:06, 7.33MB/s]
  8%|8         | 4.00M/48.5M [00:00<00:02, 15.9MB/s]
 14%|#4        | 7.00M/48.5M [00:00<00:02, 18.7MB/s]
 19%|#8        | 9.00M/48.5M [00:03<00:24, 1.72MB/s]
 23%|##2       | 11.0M/48.5M [00:04<00:20, 1.89MB/s]
 25%|##4       | 12.0M/48.5M [00:05<00:21, 1.80MB/s]
 27%|##6       | 13.0M/48.5M [00:05<00:18, 1.96MB/s]
 29%|##8       | 14.0M/48.5M [00:05<00:15, 2.32MB/s]
 31%|###       | 15.0M/48.5M [00:06<00:12, 2.82MB/s]
 35%|###5      | 17.0M/48.5M [00:06<00:07, 4.19MB/s]
 41%|####1     | 20.0M/48.5M [00:06<00:04, 6.69MB/s]
 45%|####5     | 22.0M/48.5M [00:06<00:03, 8.45MB/s]
 52%|#####1    | 25.0M/48.5M [00:06<00:02, 11.1MB/s]
 56%|#####5    | 27.0M/48.5M [00:06<00:01, 12.5MB/s]
 62%|######1   | 30.0M/48.5M [00:06<00:01, 15.1MB/s]
 68%|######8   | 33.0M/48.5M [00:07<00:00, 17.1MB/s]
 74%|########4 | 36.0M/48.5M [00:07<00:00, 18.5MB/s]
```

The dataset is opened.

Reading the Data

```
import zipfile
import csv
import pandas as pd
```

Class data set will extract and create new data .csv files.

```
class Dataset:
    def __init__(self, csv_filename):
        self.dataset = pd.read_csv(csv_filename, dtype='unicode')
```

Creating Datasets

- To create subsets of data we create definition “generate_subset.”
- When the function is called, we pass in the tablename and the column headers we want the new table to have.

```
class Dataset:
    def __init__(self, csv_filename):
        self.dataset = pd.read_csv(csv_filename, dtype='unicode')
        filecheck(csv_filename)

    def __generateMetrics(self, subset, columns):
        numberOfRows = len(subset)
        numberOfColumns = len(columns)
        output = "Number of Rows: {}\n".format(numberOfRows) + "Number of Columns: {}\n".format(numberOfColumns)
        return output

    def generate_subset(self, table_name, columns):
        test_data = self.dataset[columns]
        try:
            path = os.path.join('exported', table_name)
            test_data.to_csv(path+'.csv')
            #Check if csv file was made
            filecheck(path+'csv')
            with open(path+'_metrics.txt', 'w') as file:
                file.write(self.__generateMetrics(test_data, columns))
                #Check if the metrics file was made
                filecheck(path+'_metrics.txt')
        except Exception as e:
            logging.error(e)
            print(e)
            return
        return test_data.head()
```

Extract the Dataset

- Data is imported onto the local machine.
- Ready file for further use by extracting the file once downloaded.
- Output the data to test for successful extraction.

Divide the Data

- In order to consolidate the information, we must split the date into smaller table.
- Each table contains attributes directly associated with the entity.
- Information will be divided into seven tables for easier querying.
- Each table below will have a primary key id which will correspond to the table described above.

Transform the Data

- We must ensure that a per incident we can properly query the necessary information.
- We must first establish each unique incident based on time. This is the primary table for the entity tree. This table will be named time; of which, we will populate with each unique incident.
- Next, each incident has three sub entities; A report, an Agency, and Driver(s). It is important to note than an incident may involve multiple drivers.
- Each driver has a vehicle table, while each agency has a sub-agency that may have involved on the incident.

Creating the ER Diagram

Some columns are repeated in the dataset and can be eliminated

- Property damage
- Fatal
- Commercial license

Key

Entity
Action
Attribute

Table Driver has **Vehicle**

- *Race*
- *Gender*
- *Driver City*
- *Driver State*
- *DL State*

Table Vehicle

- *VehicleType*
- *Year*
- *Make*
- *Model*
- *Color*

Table Agency has **Table SubAgency**

- *Agency*

Table SubAgency

- *SubAgency*

Table Report

- *Accident*
- *Belts*
- *Personal Injury*
- *Property Damage*
- *Fatal*
- *Commercial License*
- *HAZMAT*

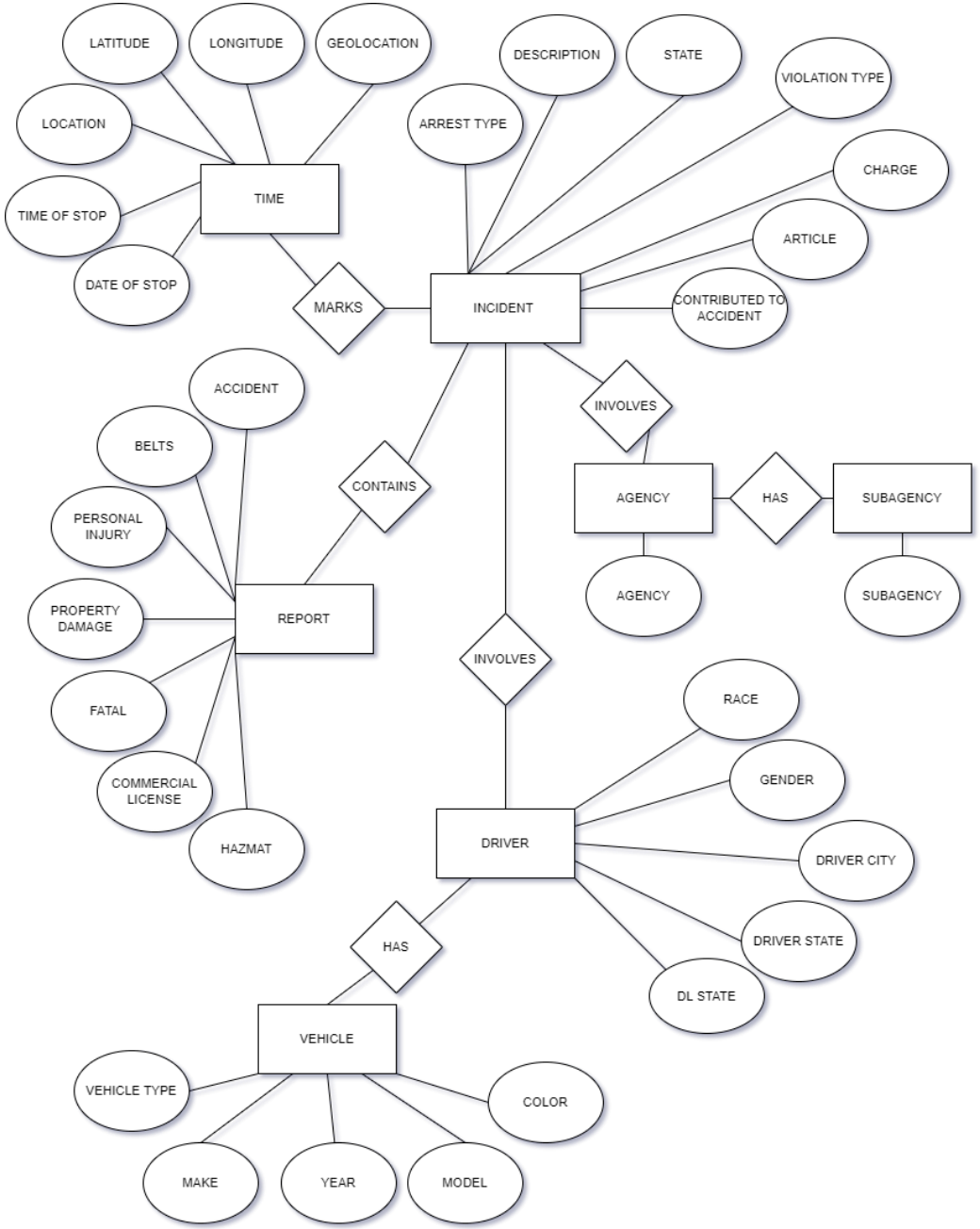
Table Incident

- *Arrest Type*
- *Description*
- *State*
- *Violation Type*
- *Charge*
- *Article*
- *Contributed To Accident*

Table TimeStamp

- *Date Of Stop*
- *Time Of Stop*
- *Location*
- *Latitude*
- *Longitude*
- *Geolocation*

ER Diagram


LATITUDE
LONGITUDE
GEOLOCATION
LOCATION
TIME OF STOP
DATE OF STOP
TIME
DESCRIPTION
STATE
VIOLATION TYPE
ARREST TYPE
CHARGE
ARTICLE
CONTRIBUTED TO ACCIDENT
MARKS
INCIDENT
ACCIDENT
BELTS
PERSONAL INJURY
PROPERTY DAMAGE
FATAL
COMMERCIAL LICENSE
HAZMAT
CONTAINS
REPORT
INVOLVES
AGENCY
HAS
SUBAGENCY
AGENCY
SUBAGENCY
INVOLVES
RACE
GENDER
DRIVER CITY
DRIVER STATE
DL STATE
DRIVER
HAS
VEHICLE
VEHICLE TYPE
COLOR
MAKE
YEAR
MODEL

Mapping with Geolocation

- In order for each table to be represented as a point on a map, we must add the Longitude and Latitude columns to each subset.
- If a geolocation is not available for a violation(row), we must omit that data from the final result.
- The final dataset will consist of each violation including data of Longitude and Latitude. This new dataset will be called Traffic_Violations_Final.csv

Mapping with Geolocation
(Continued)

[6]:
```
#Filter out data without geolocation (Longitude and Latitude)
Process_Data = pd.read_csv('Traffic_Violations.csv')
Process_Data = Process_Data.dropna(subset=['Longitude', 'Latitude'])

#Process_Data = Process_Data[Process_Data['State'].isin(['WA']) ]

#Save dataframe as new csv
Process_Data.to_csv(pathlib.Path(".", "Traffic_Violations_Final.csv").resolve())
#Process_Data.to_csv(r'C:\Users\KEVAL\Desktop\DataEngineering_CapstoneProject\Traffic_Violations_Final.csv')

#Show Process_Data
display(Process_Data)
```

Logging Errors

- Create a log file to store log information.
- Create a exception to catch errors and log them in the log file.

```
import os
import logging
from os.path import exists
```

```
if not os.path.exists('exported'):
    os.makedirs('exported')
```

```
#Logging File for Metrics
logging.basicConfig(filename='logs.log', level=logging.ERROR)
```

```
class Dataset:
    def __init__(self, csv_filename):
        self.dataset = pd.read_csv(csv_filename, dtype='unicode')
        filecheck(csv_filename)

    def __generateMetrics(self, subset, columns):
        numberOfRows = len(subset)
        numberOfColumns = len(columns)
        output = "Number of Rows: {}\n".format(numberOfRows) + "Number of Columns: {}\n".format(numberOfColumns)
        return output

    def generate_subset(self, table_name, columns):
        test_data = self.dataset[columns]
        try:
            path = os.path.join('exported', table_name)
            test_data.to_csv(path+'.csv')
            #Check if csv file was made
            filecheck(path+'csv')
            with open(path+'_metrics.txt', 'w') as file:
                file.write(self.__generateMetrics(test_data, columns))
                #Check if the metrics file was made
                filecheck(path+'_metrics.txt')
        except Exception as e:
            logging.error(e)
            print(e)
            return
        return test_data.head()
```

Logging Metrics

- Create a definition "__generateMetrics".
- To create the structure to document table information.

- Write to file metrics information.

```
class Dataset:
    def __init__(self, csv_filename):
        self.dataset = pd.read_csv(csv_filename, dtype='unicode')
        filecheck(csv_filename)

    def __generateMetrics(self, subset, columns):
        numberOfRows = len(subset)
        numberOfColumns = len(columns)
        output = "Number of Rows: {}\n".format(numberOfRows) + "Number of Columns: {}\n".format(numberOfColumns)
        return output

    def generate_subset(self, table_name, columns):
        test_data = self.dataset[columns]
        try:
            path = os.path.join('exported', table_name)
            test_data.to_csv(path+'.csv')
            #Check if csv file was made
            filecheck(path+'csv')
            with open(path+'_metrics.txt', 'w') as file:
                file.write(self.__generateMetrics(test_data, columns))
                #Check if the metrics file was made
                filecheck(path+'_metrics.txt')
        except Exception as e:
            logging.error(e)
            print(e)
            return
        return test_data.head()
```

Unit Testing

Check file creation

- Create new file to store testing data.
- Check if file was created.
- Print Errors to UnitTest.txt file.

```
#Logging File for Metrics
logging.basicConfig(filename='logs.log', level=logging.ERROR)

#Text File for Unit Tests
UnitTest = open(pathlib.Path(".", "UnitTest.txt").resolve(),"w")
#UnitTest = open(r"C:\Users\KEVAL\Desktop\DataEngineering_CapstoneProject\UnitTest.txt", "w")

UnitTest.write("This is the UNIT Test File"+ "\n")
```

```
#Check if file is in folder

def filecheck(location):
    # check if it exist
    try:
        file_exists = os.path.exists(location)
        path = os.path.dirname(os.path.abspath(location))
        #Print to UnitTest
        UnitTest.write("Successfull!! -- The path of the file is: " + path + "-->" + location + "\n")
    except Exception as e:
        #Print to UnitTest.txt
        UnitTest.write("ERROR: Unsuccessfull!! -- The path of the file is: " + path + "-->" + location + "\n")
        print(e)
        print("Error: File NOT IN LOCATION " + location)
        return
    return
```

Unit Testing
(continued)

- Call definition “filecheck” to check conditions for Unit Test.

```
class Dataset:
    def __init__(self, csv_filename):
        self.dataset = pd.read_csv(csv_filename, dtype='unicode')
        filecheck(csv_filename)

    def __generateMetrics(self, subset, columns):
        numberOfRows = len(subset)
        numberOfColumns = len(columns)
        output = "Number of Rows: {}\n".format(numberOfRows) + "Number of Columns: {}\n".format(numberOfColumns)
        return output

    def generate_subset(self, table_name, columns):
        test_data = self.dataset[columns]
        try:
            path = os.path.join('exported', table_name)
            test_data.to_csv(path+'.csv')
            #Check if csv file was made
            filecheck(path+'csv')
            with open(path+'_metrics.txt', 'w') as file:
                file.write(self.__generateMetrics(test_data, columns))
                #Check if the metrics file was made
                filecheck(path+'_metrics.txt')
        except Exception as e:
            logging.error(e)
            print(e)
            return
        return test_data.head()
```

Subset Results

```
traffic_data = Dataset("Traffic_Violations_Final.csv")
```

```
traffic_data.generate_subset('Incident', ['Arrest Type', 'Description', 'State', 'Violation Type', 'Charge', 'Article', 'Cont
```

]:

	Arrest Type	Description	State	Violation Type	Charge	Article	Contributed To Accident	Longitude	Latitude
0	A - Marked Patrol	DRIVING WHILE IMPAIRED BY ALCOHOL	MD	Citation	21-902(b1)	Transportation Article	No	-77.09310515	38.9835782
1	A - Marked Patrol	FAILURE TO STOP AT STOP SIGN	MD	Citation	21-707(a)	Transportation Article	No	-77.25358095	39.1618098166667
2	A - Marked Patrol	DRIVER USING HANDS TO USE HANDHELD TELEPHONE W...	VA	Citation	21-1124.2(d2)	Transportation Article	No	-77.1007551666667	38.9827307333333
3	A - Marked Patrol	FAILURE STOP AND YIELD AT THRU HWY	MD	Citation	21-403(b)	Transportation Article	No	-77.2290883333333	39.1628883333333
4	A - Marked Patrol	OCCUPANT UNDER 16 NOT RESTRAINED BY SEATBELT	MD	Citation	22-412.3(b)	Transportation Article	No	-76.9696780666667	39.06914295

```
traffic_data.generate_subset('Report', ['HAZMAT', 'Commercial License', 'Fatal', 'Property Damage', 'Personal Injury', 'Belts
```

]:

	HAZMAT	Commercial License	Fatal	Property Damage	Personal Injury	Belts	Accident	Longitude	Latitude
0	No	No	No	No	No	No	No	-77.09310515	38.9835782
1	No	No	No	No	No	No	No	-77.25358095	39.1618098166667
2	No	No	No	No	No	No	No	-77.1007551666667	38.9827307333333
3	No	No	No	Yes	No	No	No	-77.2290883333333	39.1628883333333
4	No	No	No	No	No	No	No	-76.9696780666667	39.06914295

Subset Results

(Continued)

```
traffic_data.generate_subset('Time', ['Date Of Stop', 'Time Of Stop', 'Location', 'Latitude', 'Longitude', 'Geolocation'])
```

]:

	Date Of Stop	Time Of Stop	Location	Latitude	Longitude	Geolocation
0	12/20/2012	00:41:00	NORFOLK AVE / ST ELMO AVE	38.9835782	-77.09310515	(38.9835782, -77.09310515)
1	07/20/2012	23:12:00	WISTERIA DR @ WARING STATION RD	39.1618098166667	-77.25358095	(39.1618098166667, -77.25358095)
2	03/19/2012	16:10:00	CLARENDON RD @ ELM ST. N/	38.9827307333333	-77.1007551666667	(38.9827307333333, -77.1007551666667)
3	12/01/2014	12:52:00	CHRISTOPHER AVE/MONTGOMERY VILLAGE AVE	39.1628883333333	-77.2290883333333	(39.1628883333333, -77.2290883333333)
4	06/09/2012	21:19:00	2068 HARLEQUIN TERRACE	39.06914295	-76.9696780666667	(39.06914295, -76.9696780666667)

```
traffic_data.generate_subset('Agency', ['Agency','Longitude', 'Latitude'])
```

]:

	Agency	Longitude	Latitude
0	MCP	-77.09310515	38.9835782
1	MCP	-77.25358095	39.1618098166667
2	MCP	-77.1007551666667	38.9827307333333
3	MCP	-77.2290883333333	39.1628883333333
4	MCP	-76.9696780666667	39.06914295

Subset Results
(Continued)

```
traffic_data.generate_subset('Subagency', ['SubAgency','Longitude', 'Latitude'])
```

]:

	SubAgency	Longitude	Latitude
0	2nd district, Bethesda	-77.09310515	38.9835782
1	5th district, Germantown	-77.25358095	39.1618098166667
2	2nd district, Bethesda	-77.1007551666667	38.9827307333333
3	6th district, Gaithersburg / Montgomery Village	-77.2290883333333	39.1628883333333
4	3rd district, Silver Spring	-76.9696780666667	39.06914295

```
traffic_data.generate_subset('Driver', ['DL State', 'Driver State', 'Driver City', 'Gender', 'Race','Longitude', 'Latitude'])
```

]:

	DL State	Driver State	Driver City	Gender	Race	Longitude	Latitude
0	MD	MD	DERWOOD	M	WHITE	-77.09310515	38.9835782
1	MD	MD	GERMANTOWN	F	ASIAN	-77.25358095	39.1618098166667
2	VA	VA	ARLINGTON	M	HISPANIC	-77.1007551666667	38.9827307333333
3	MD	MD	UPPER MARLBORO	F	BLACK	-77.2290883333333	39.1628883333333
4	MD	MD	SILVER SPRING	F	WHITE	-76.9696780666667	39.06914295

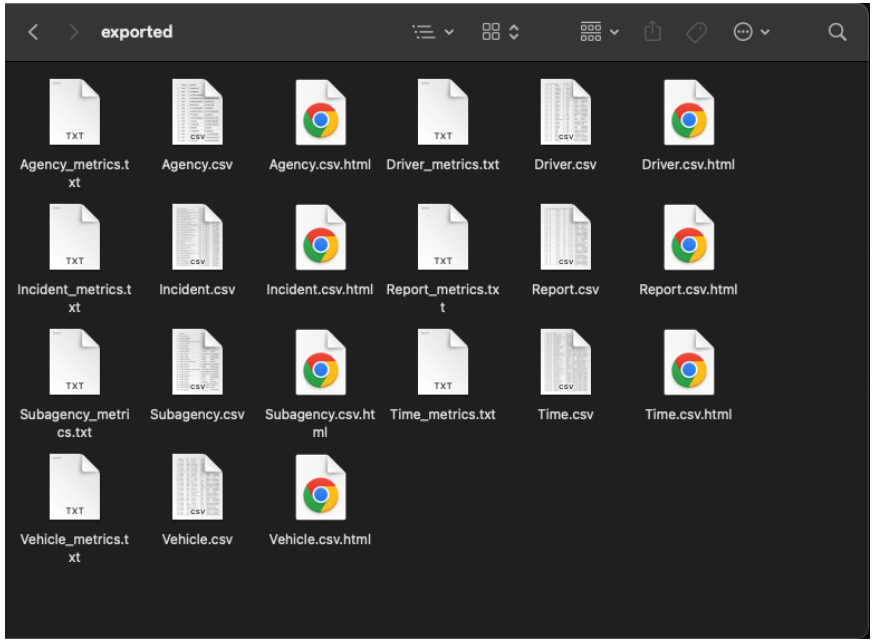
```
traffic_data.generate_subset('Vehicle', ['VehicleType', 'Make', 'Year', 'Model', 'Color','Longitude', 'Latitude'])
```

]:

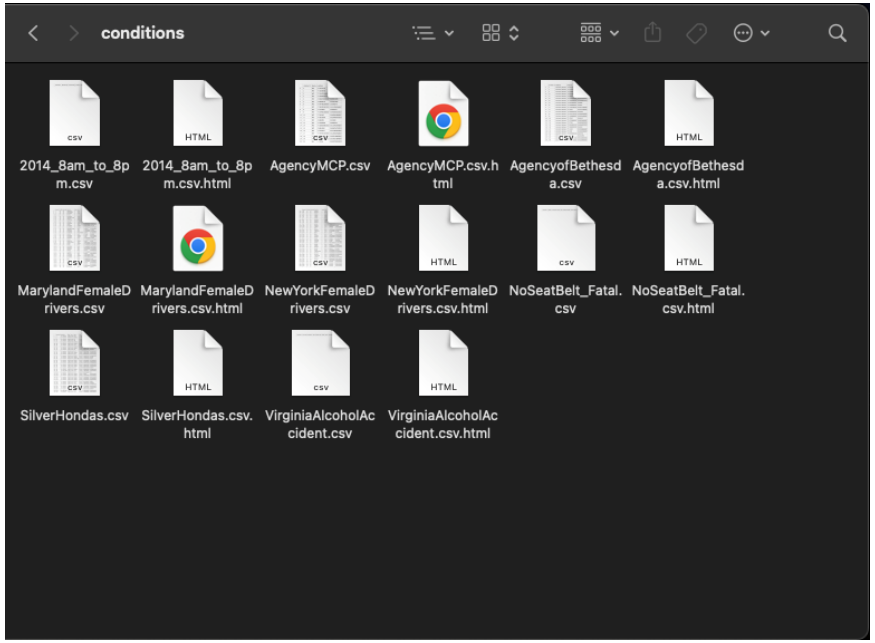
	VehicleType	Make	Year	Model	Color	Longitude	Latitude
0	02 - Automobile	AUDI	2005.0	4S	GRAY	-77.09310515	38.9835782
1	02 - Automobile	TOYT	2002.0	4S	RED	-77.25358095	39.1618098166667
2	02 - Automobile	HONDA	1996.0	CIVIC	SILVER	-77.1007551666667	38.9827307333333
3	02 - Automobile	HONDA	2001.0	ACCORD	SILVER	-77.2290883333333	39.1628883333333
4	02 - Automobile	CHEVROLET	2004.0	IMPALA	SILVER	-76.9696780666667	39.06914295

Subset Files

- The subsets generated .csv files which are stored in the folder “exported”.



- Later we use this function in combination with geomapping to create subsets of .csv files that are stored In the folder “conditions”.



Geomapping the Data

Definition “Plotting” will create a map of given .csv data file.

A data frame of the map is first created. We use the longitude and latitude.

A map of the US is the first layer of our map of which ontop we will place our data points.

The Resulting file is a .html file that can be opened on the web browser with zooming functionality.

```
#Plotting the Data
```

```
# Takes in File, Plots Data, and Saves Result File
class Plotting:

    def __init__(self, csv_filename):

        file = csv_filename

        #Direct Path
        data_path = pathlib.Path(".", file).resolve()
        df = pd.read_csv(data_path, engine = 'python')

        #Folder path
        #data_path = pathlib.Path("./exported", file).resolve()
        #df = pd.read_csv(data_path, engine = 'python')
        self.CreateMap(df, file)
        return

    #Map the dataframe
    def CreateMap(self, dataframe, filename):
        file = filename
        df_geo = gdp.GeoDataFrame(dataframe, geometry = gdp.points_from_xy(dataframe.Longitude,dataframe.Latitude))

        try:

            #Plot Map of USA with Cities
            us_cities = pd.read_csv("https://raw.githubusercontent.com/plotly/datasets/master/us-cities-top-1k.csv")
            #fig = px.scatter_mapbox(us_cities, lat="lat", lon="lon", color_discrete_sequence=["fuchsia"], zoom=3, height=500)

            #Data Plot
            fig = px.scatter_mapbox(df_geo,lat='Latitude', lon='Longitude', title="Map of Violations", color_discrete_sequence=["fuchsia"], height=750)
            fig.update_layout(mapbox_style="open-street-map")
            fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})

            #Write Data to file ---- NEEDS TO PRINT WITH TABLE NAME
            pio.write_html(fig,file+".html")
            #pio.write_html(fig,"figure/fig1.html")

            filecheck(file+'.html')

            #Create Unique File for each query
            #path = os.path.join('figure', table_name)
            #fig.to_html(path+'.html')

        except Exception as e:
            logging.error(e)
            print(e)
            return

        return
```

Creating Reports

Here we map the the original dataset as well as the subsets we created above.

Because the size of the original dataset is too large, viewing the .html file may not work. We must break down the data into smaller chunks that we can work with.

```
Plotting("Traffic_Violations_Final.csv")
Plotting("./exported/Incident.csv")
Plotting("./exported/Report.csv")
Plotting("./exported/Time.csv")
Plotting("./exported/Agency.csv")
Plotting("./exported/Subagency.csv")
Plotting("./exported/Driver.csv")
Plotting("./exported/Vehicle.csv")
```

Conditional Queries

Defintion condplot takes in the:

- location which is the the path to the .csv file containing the original set of data.
- title which is the name for the new plot .html based on the conditions.
- condition which is the conditional requirements for the mapping.

A new subset .csv file containing only the data needed for the mapping.

Plotting takes in the new .csv file and generates a new .html.

```
####Plotting Conditional Data
def condplot(location, title, condition):
    df = pd.read_csv(location, dtype='unicode')
    df = eval(condition)

    # write output
    path = os.path.join('conditions', title)
    df.to_csv(path+'.csv')

    #Create and Save Graph
    Plotting("./conditions/"+title+".csv")
```

Custom Queries

Here are some examples of conditional queries that were geomapped using condplot().

```
#Only Impared by alcohol in Virginia who contributed to accident
condplot("./exported/Incident.csv", "VirginiaAlcoholAccident", "df[(df['Description'].str.contains('Alcohol')) & (df['State'] =='VA')& (df['Contributed To Accident'] =='YES')]" )
```

```
#Only Fatal with property damage not wearing a seatbelt
condplot("./exported/Report.csv", "NoSeatBelt_Fatal", "df[(df['Fatal'] == 'YES') & (df['Property Damage'] =='YES') & (df['Belts'] =='NO') ]" )
```

```
#Only 2014 between 8am and 5pm
condplot("./exported/Time.csv", "2014_8am_to_8pm", "df[(df['Date Of Stop'] == '2014') & (df['Time Of Stop'] >'08:00:00') & (df['Time Of Stop'] <'17:00:00') ]" )
```

```
#Only vehicles that are honda and silver
condplot("./exported/Agency.csv", "AgencyMCP", "df[(df['Agency'] == 'MCP')]" )
```

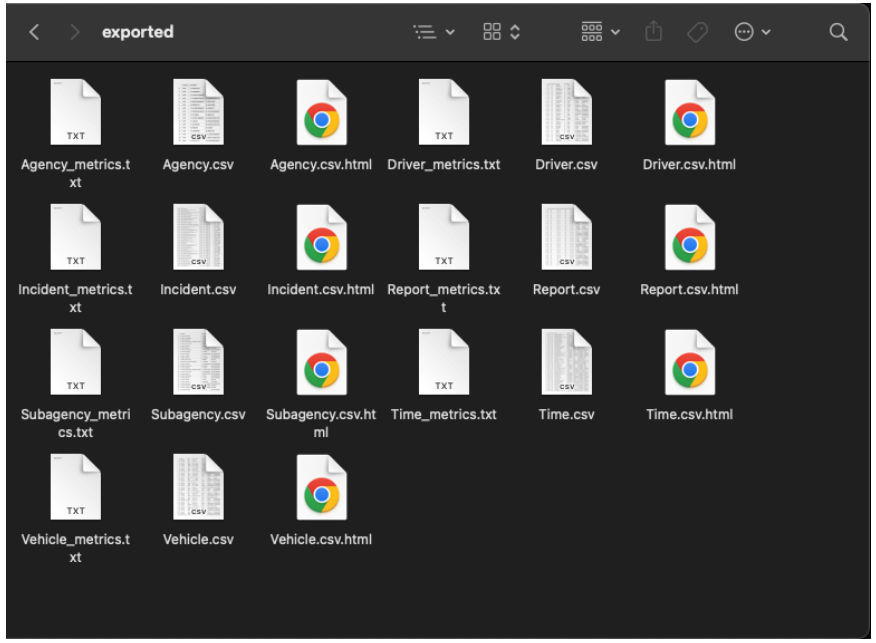
```
#Only 2nd district Bethesda
condplot("./exported/Subagency.csv", "AgencyofBethesda", "df[(df['SubAgency'] == '2nd district, Bethesda')]" )
```

```
#Only Females with New York Drivers License
condplot("./exported/Driver.csv", "NewYorkFemaleDrivers", "df[(df['Driver State'] == 'NY') & (df['Gender'] =='F')]" )
```

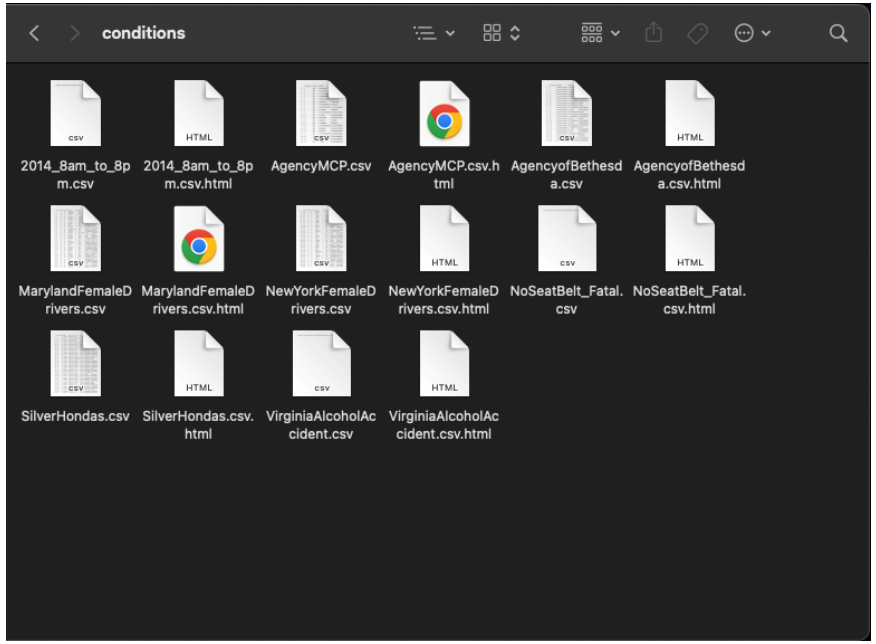
```
#Only vehicles that are honda and silver
condplot("./exported/Vehicle.csv", "SilverHondas", "df[(df['Make'] == 'HONDA') & (df['Color'] =='SILVER')]" )
```

Mapping Result Files

- The geomapping generated .html files which are stored in the folder “exported”.



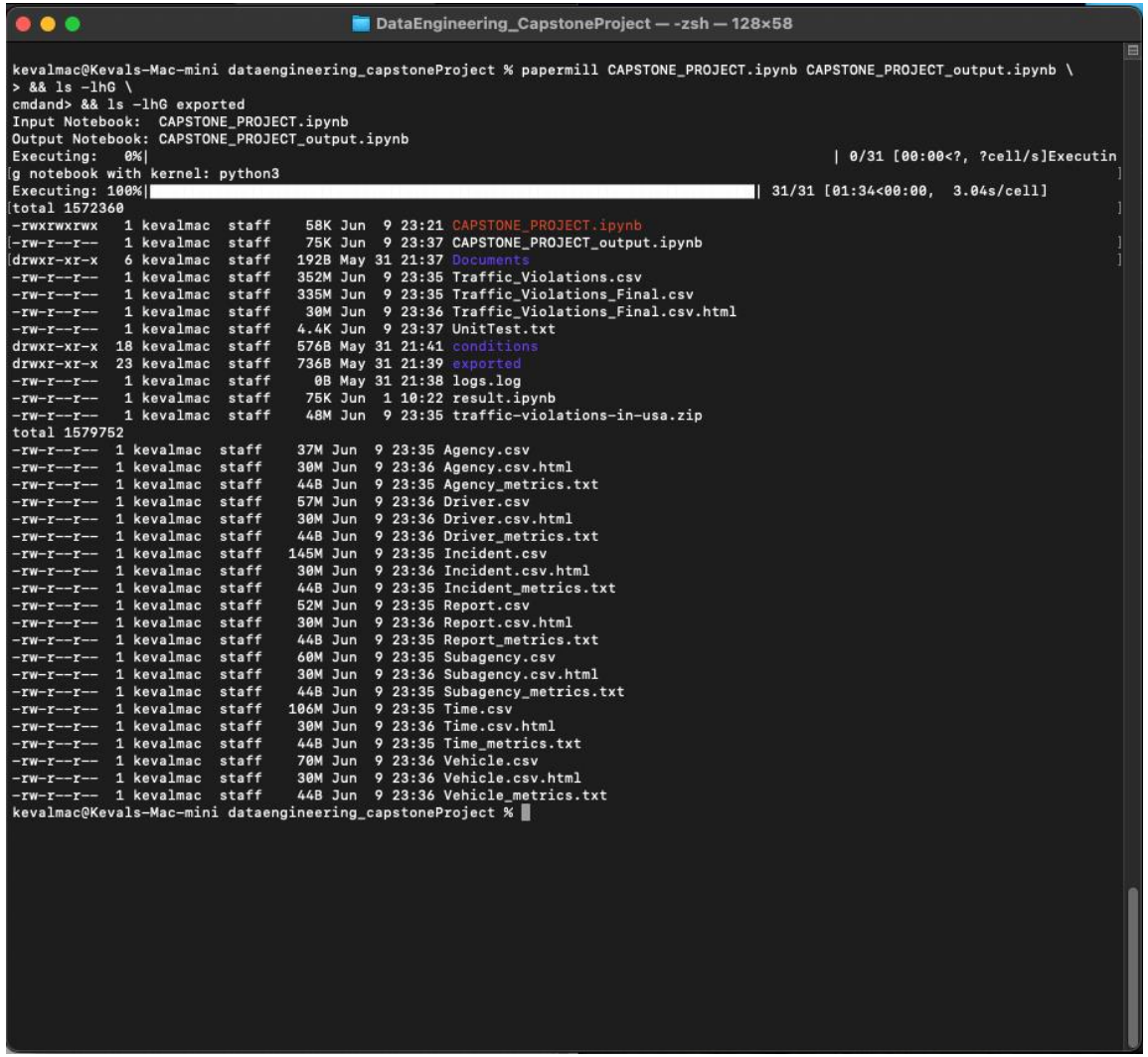
- Subset data with conditions are stored in the folder “conditions,” as .html files with the filename reflecting the initial conditions.



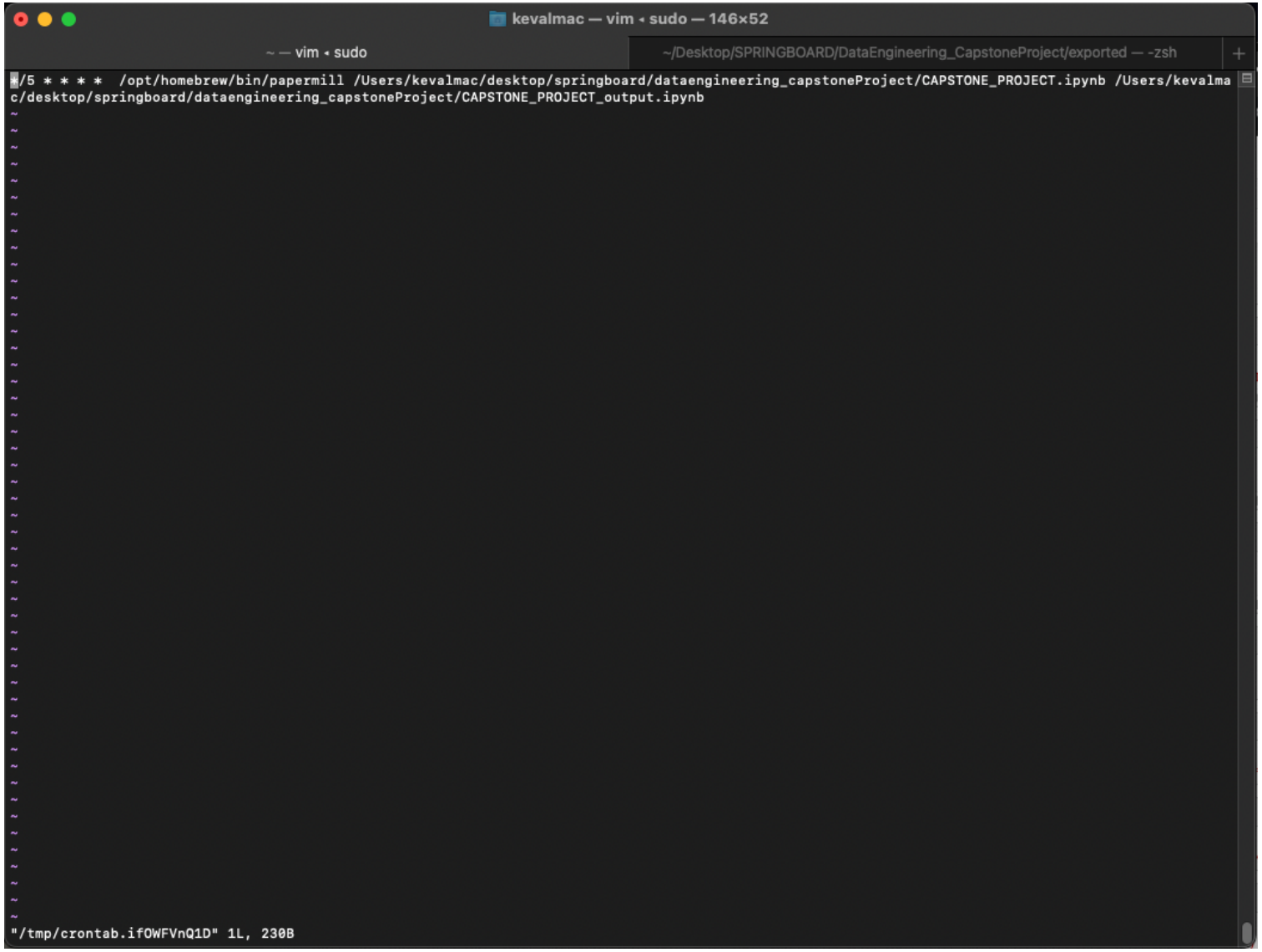
Final Program

- Refer to Capstone_Project.pdf which can be found in the documents folder.

Scheduling with Cronjob using Papermill



Cronjob Configuration



Running the Cronjob