# Data Ingestion:

## Shell Commands:

Commands:

- For downloading the dataset
    - curl -o motorVehicleCollisions https://data.cityofnewyork.us/api/views/h9gi-nx95/rows.csv?date=20231109&accessType=DOWNLOAD
- Put the dataset in hadoop directory
    - hadoop fs -mkdir project
    - hadoop fs -put motorVehicleCollisions project
- Run the shell script run.sh which contains all the commands for compiling and running the hadoop jobs. The commands are:
    - hadoop fs -rm -r project/output
    - javac -classpath $(hadoop classpath) *.java
    - jar cvf dataClean.jar *.class
    - hadoop jar dataClean.jar DataCleaning project/motorVehicleCollisions project/output
    - hadoop fs -get project/output/part-r-00000
    - hadoop fs -get project/output/data_profiling_count/part-r-00000
    - hadoop fs -get project/output/data_profiling_value/part-r-00000

Output files:

- **project/output/part-r-00000** - contains the output of the dataset after data cleaning
- **project/output/data_profiling_count/part-r-00000** - contains the output of data profiling of columns, specifically for counting operations
- **project/output/data_profiling_value/part-r-00000** - contains the output of data profiling of columns, specifically for finding min/max operations

## Data Source:

### Motor Vehicle Collisions - Crashes Dataset:

The Motor Vehicle Collisions crash table contains details on the crash event. Each row represents a crash event. The Motor Vehicle Collisions data tables contain information from all police reported motor vehicle collisions in NYC. The data in this table goes back to April 2016 when crash reporting switched to an electronic system. The dataset can be downloaded in form of a csv file.

Columns in the dataset:

- CRASH DATE
- CRASH TIME
- BOROUGH
- ZIP CODE
- LATITUDE
- LONGITUDE
- LOCATION
- ON STREET NAME
- CROSS STREET NAME
- OFF STREET NAME
- NUMBER OF PERSONS INJURED
- NUMBER OF PERSONS KILLED
- NUMBER OF PEDESTRIANS INJURED
- NUMBER OF PEDESTRIANS KILLED
- NUMBER OF CYCLIST INJURED
- NUMBER OF CYCLIST KILLED
- NUMBER OF MOTORIST INJURED
- NUMBER OF MOTORIST KILLED
- CONTRIBUTING FACTOR VEHICLE 1
- CONTRIBUTING FACTOR VEHICLE 2
- CONTRIBUTING FACTOR VEHICLE 3
- CONTRIBUTING FACTOR VEHICLE 4
- CONTRIBUTING FACTOR VEHICLE 5
- COLLISION_ID
- VEHICLE TYPE CODE 1
- VEHICLE TYPE CODE 2
- VEHICLE TYPE CODE 3
- VEHICLE TYPE CODE 4
- VEHICLE TYPE CODE 5

# Data Cleaning:

In the data cleaning part, I did the following operations:

1. **Removed the CSV Header:**
   The initial dataset contained a header row that provided column names. To ensure consistency and avoid issues during analysis, the header was removed.
   This is because headers can be redundant when processing data in a Hadoop MapReduce environment, and removing them ensures that the analysis is not impacted by the presence of an extra row.
2. **Remove malformed csv rows:**
   Identified and removed rows that did not conform to the expected structure of a CSV,

which might include rows with missing or extra values. There are 29 columns in the dataset and thus each row in the dataset should contain 29 comma separated values (the values can be empty too). There are some rows in the dataset which have less/more than 29 columns. These rows were removed in the data cleaning process. Malformed rows can lead to errors during analysis. Thus, removing them ensures that the dataset is clean and adheres to the expected format.

3. **Replace NaN values of numerical columns with 0:**
   Imputed missing values (NaN) in numerical columns with the value 0. By running the data profiling code, I found out that there are some columns which are missing values. For example, the columns: BOROUGH, ZIP CODE, NUMBER OF PERSONS INJURED, NUMBER OF PERSONS KILLED etc, were missing values. Since, only the columns NUMBER OF PERSONS INJURED, NUMBER OF PERSONS KILLED contained numerical values, only their NaN values could be replaced with 0. I also verified in this part that the important columns are not missing any entries.
   NaN values can affect the analysis of numerical data. Replacing them with 0 allows for a more consistent dataset and prevents potential issues during mathematical operations.
4. **Processed the street names to make them homogeneous:**
   Standardized the format of street names to ensure homogeneity. This included converting names to lowercase, removing leading or trailing whitespaces, also removing unneeded tabs spaces in the street names and applying other transformations. Entries for the same street can differ in different entries, for example, ‘avenue’ in street names can be written as ‘ave’ or ‘av’. Thus for the street names, I followed the following mapping: Map (“STREET” : “ST”, “AVENUE” : “AVE”, “AV” : “AVE”, “BOULEVARD”: “BLVD”, “ROAD”: “RD”, “PARKWAY”: “PKY”, “HIGHWAY”: “HWY”, “COURT”: “CT”, “PLACE”: “PL”, “SQUARE”: “SQ”, “TURNPIKE”: “TPKE”, “LANE”: “LN”, “POINT”: “PT”, “PLAZA”: “PZ”).
   Homogeneous street names facilitate efficient analysis and help avoid discrepancies due to variations in naming conventions. This is important for our project because the street name is the common factor which can be used to join the different datasets in our group project for combined analysis of different datasets.
5. **Removed unnecessary columns:**
   In this, I identified and removed columns that were deemed unnecessary or redundant for the analysis. For example, the column LOCATION contained a comma separated latitude and longitude of the place. This is redundant as we already have different columns LATITUDE & LONGITUDE for the same. Thus, I deleted this column to reduce space usage.
   Removing unnecessary columns reduces the dimensionality of the dataset, making it more manageable and improving processing efficiency.

## Data Profiling:

In this part, I conducted a detailed analysis of each column, providing additional information such as data types, unique values, and potential data quality issues. Understanding the

characteristics of each column is essential for meaningful analysis. This step enhances the overall data profiling process.

1. **Count the number of missing values for each of the columns:**
   This operation is crucial for understanding the completeness of the dataset. By counting the number of missing values in each column, we can identify which columns have incomplete data. This information was used in data cleaning, such as deciding whether to remove, impute, or ignore the missing values. Below is the table of missig values for each of the columns.

| Column Name | Missing Values Count |
|---|---|
| CRASH DATE | 0 |
| CRASH TIME | 0 |
| BOROUGH | 636731 |
| ZIP CODE | 636976 |
| LATITUDE | 231823 |
| LONGITUDE | 231823 |
| LOCATION | 231823 |
| ON STREET NAME | 432625 |
| CROSS STREET NAME | 769841 |
| OFF STREET NAME | 1706502 |
| NUMBER OF PERSONS INJURED | 18 |
| NUMBER OF PERSONS KILLED | 31 |
| NUMBER OF PEDESTRIANS INJURED | 0 |
| NUMBER OF PEDESTRIANS KILLED | 0 |
| NUMBER OF CYCLIST INJURED | 0 |
| NUMBER OF CYCLIST KILLED | 0 |
| NUMBER OF MOTORIST INJURED | 0 |
| NUMBER OF MOTORIST KILLED | 0 |
| CONTRIBUTING FACTOR VEHICLE 1 | 6562 |
| CONTRIBUTING FACTOR VEHICLE 2 | 314520 |
| CONTRIBUTING FACTOR VEHICLE 3 | 1901089 |
| CONTRIBUTING FACTOR VEHICLE 4 | 2013993 |
| CONTRIBUTING FACTOR VEHICLE 5 | 2037891 |
| COLLISION_ID | 0 |
| VEHICLE TYPE CODE 1 | 13173 |
| VEHICLE TYPE CODE 2 | 386452 |

| | |
|---|---|
| VEHICLE TYPE CODE 3 | 1906273 |
| VEHICLE TYPE CODE 4 | 2015108 |
| VEHICLE TYPE CODE 5 | 2038160 |

2. **Count the number of accidents over year/month:**
   This operation can reveal temporal trends in the data. For example, it can show whether accidents are more common at certain times of the year or whether the number of accidents is increasing or decreasing over time. Also, this can be used to figure out if there are no major missing parts in the data, as the counts should not rapidly change over years and months.
3. **Count the number of people injured/killed over year/month:**
   Similar to the previous point, this operation can show temporal trends in the severity of accidents. It can reveal whether accidents are becoming more or less severe over time.
4. **Count the number of accidents per borough:**
   This operation can reveal geographical trends in the data. It can show which boroughs have the most accidents and may need more attention in terms of traffic safety measures. Also, was used to verify that all the boroughs were included in the dataset and data is not missing for a particular borough.
5. **Count the number of people injured/killed over year/month per borough:**
   This operation combines temporal and geographical trends. It can show how the severity of accidents in each borough changes over time. Also, to verify that there are no major missing parts in the data.
6. **Find the min/max number of people injured/killed in an accident:**
   This operation can provide insights into the range of accident severity. The minimum and maximum number of people injured or killed in an accident can give a sense of the best-case and worst-case scenarios. Also, it is used to verify if there are any outliers in the data and as such if any outlier detection and removal is needed.
7. **Find the min/max number of people injured/killed in an accident over year/month per borough:**
   This operation can provide more detailed insights into accident severity in each borough over time. It can show the best-case and worst-case scenarios for each borough in each month or year. Also, similarly to above, it is used to verify if there are any outliers in the data and as such if any outlier detection and removal is needed.

## Java Classes Used:

1. **DataCleaningMapper & DataCleaningReducer:**
   This MapReduce program is responsible for cleaning the data. The DataCleaningMapper is parsing the input data and filtering out or correcting any inconsistencies, errors, or missing values. The DataCleaningReducer is aggregating the cleaned data and further filtering the data based on certain conditions.

2. **DataProfilingCountMapper & DataProfilingCountReducer:**
   This MapReduce program is responsible for profiling the data by counting certain attributes. The DataProfilingCountMapper is emitting key-value pairs where the key is the attribute to count and the value is 1. The DataProfilingCountReducer would then sum up these values for each key, resulting in a count of each attribute.
3. **DataProfilingValueMapper & DataProfilingValueReducer:**
   This MapReduce program is responsible for profiling the data by calculating min/max values. The DataProfilingValueMapper is emitting key-value pairs where the key is the attribute to calculate and the value is the attribute's value. The DataProfilingValueReducer then performs some calculation on these values for each key, such as finding the minimum, maximum, average, or total.