# **CIS 4400**

# **Project Proposal**

# **NYC Car Collisions Data Warehouse**

# **Mohamed Abouregila**

# 

# **MOHAMED ABOUREGILA**

[**mohamed.abouregila@baruchmail.cuny.edu**](mailto:jatinder.singh2@baruchmail.cuny.edu)

This project was developed under the supervision of Professor Emily Mazo as part of the fall 2020 Data Warehousing class.

Jatinder Singh contributed with the discussion of the business problem to the project and Proposed the NYC Open APIs as one of the data sources of this project.

## **The Business/Organization & Opportunity**

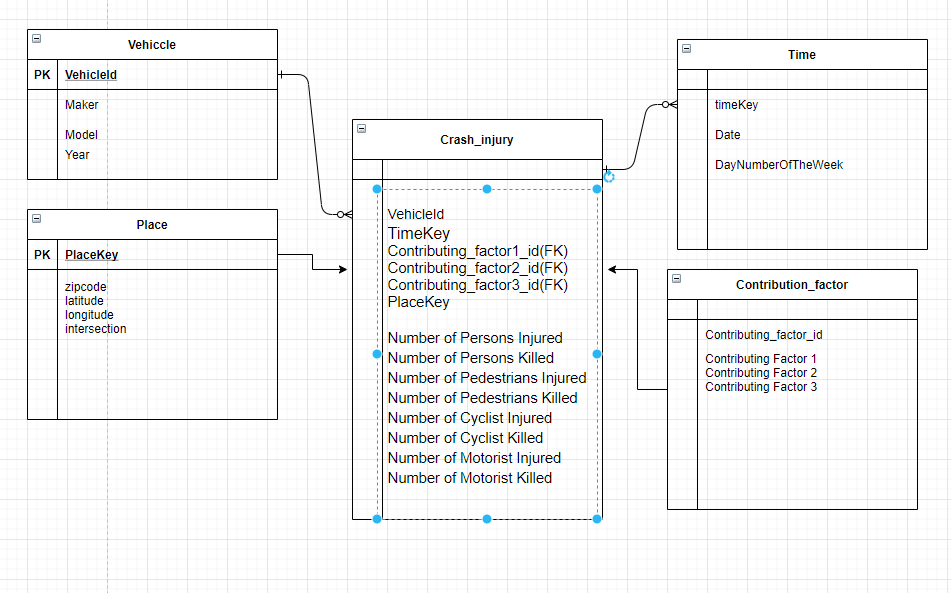
1. We are a data Engineering, and analytics contractor that helps the NYC DMV Commissioner's office to understand Civilian Car Crashes and implement policies which will make the pedestrians and users of NYC streets safer. By creating a Data Warehouse built upon various sources we look to understand which cars, groups, and areas of our city incur the most crashes and from there develop a plan to decrease that number.

## 

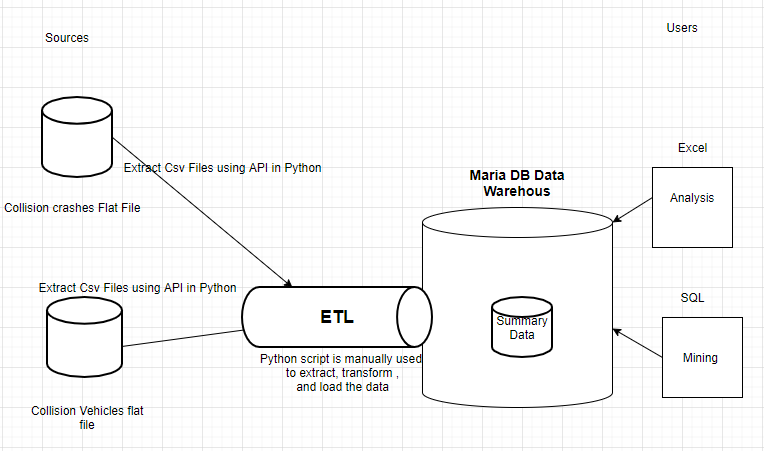
### **Problem**

1. Currently we find there to be a gap in understanding the causes of why there are over 600 car crashes throughout NYC each day. The lack of available data warehouses that have the required data that helps researcher, and analysts understand the causes of this great number of crashes makes this gap even bigger. understanding how certain times of day, races, locations, vehicle types and other constraints affect this number we look to discover a solution which will decrease the number of crashes by understanding the impact of the prior mentioned variables. The goal of this data warehouse is to be a data source for data analysts and data scientists to be able to create, visualize, and present trends and solutions based on their analysis.

### **Logical Data Model**



### **Architecture Diagram**



1. We have used these data sources for our project :
   1. NYC Open Data:
      1. Collision Crashes Flat file available on : <https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Crashes/h9gi-nx95>
      2. Collision Vehicles flat file available on: <https://data.cityofnewyork.us/Public-Safety/Motor-Vehicle-Collisions-Vehicles/bm4k-52h4>

* We are planning to use the following data sources in the next version:
  + 1. Leading causes of Death flat file available on: <https://data.cityofnewyork.us/Health/New-York-City-Leading-Causes-of-Death/jb7j-dtam>
    2. Demographic by Zip Code flat file available on: <https://data.cityofnewyork.us/City-Government/Demographic-Statistics-By-Zip-Code/kku6-nxdu>
  1. Vehicle API available at NHTSA: <https://vpic.nhtsa.dot.gov/api/>

1. We will imported the data using python scripts run on our laptops to clean the data, merge data sources, and upload the merged (transformed) data into our data warehouse
2. We are running our data warehouse on our local machines using MariaDB database
3. We are using Excel,and SQL, to build a dashboard of graphs, to visualize our KPIs.

### **Detailed Design:**

* We chose NYC open data Collision crashes flat file,Collision Vehicles flat file, because They are open source , and they have all, and the most accurate data about crashes in NYC, we also included Leading causes of Death, Demographic by Zip Code flat file so that the data warehouse could be used to analyze the accidents across different groups, and the accidents as death cause compared to other death causes. We believe the We included Vehicle API data so that the analysts could be able to investigate any relationships between car makers, models, and accidents
* In order to enhance the data warehouse performance, we have chosen to model our data warehouse using dimensional modeling.
* Looking at our KPIs (crashes per zip code, number of deaths caused by accidents per zip code) we normalized the crash\_injury\_death fact table and denormalized other dimensions like time, place, and vehicle.
* We have used Python scripts to import the data. We chose Python, because the team is familiar with it, and to reduce the cost.
* We have used Excel to build a dashboard of graphs, because the team is familiar with excel
* We haven’t considered other data analytics tools for building the dashboard like Tableau, because it's not open source, and also to reduce the time, and cost needed to train the team on these tools.

1. **Ethical issues taken into consideration**

We decided our audience are City officials who are concerned about accidents in NYC

We are concerned that Car advertisers might use our data warehouse to target specific races/groups, and exclude other races/groups in their advertisements.

In the current version the race of the drivers are not included in the data warehouse. However, we plan to include it in the next version, and will try to implement the following:

* + - 1. We are thinking about encrypting/ hashing the data related to race, groups, one way encryption the same way passwords are encrypted.
      2. We will try to develop rules to prevent querying data related to cars used by specific groups to prevent cars advertisers from using our database to discriminate against specific groups.

Disinformation:

We believe our product doesn’t have the ability to influence people’s world views, and it couldn’t be used spread disinformation, because all the data will be extracted from governmental agencies

Technical Documentation:

Code description :

The code provided in the code section extract the data, create keys when needed, and lod it to mariaDB data warehouse.

**Extracting the Data**

We use NYC Open data APIs to extract csv files. We don’t keep a copy of the extracted files, because we are facing a memory issue at this point of time. We only keep the copies temporarily on the ram, manipulate them and load to the Maria db data warehouse.

We don’t extract the whole files, but only the columns that we are interested in analyzing. If you are interested in any other column, please pass the name of the column in the url.

For Example: in the following line we only extract the zipcode and the collision\_id. If you would like to add one more column. You can add , and the column name at the end of the url:



We use python script to send a request to extract the data for every dimension. This might not be the most efficient way. However, it’s an easy way to complete the ETL that we will consider changing in later versions.

We create contributing factor\_id (surrogate key) based on the index. Other surrogate keys are created based on columns within the dimensions.

**Installation Instruction:**

If you would like to reuse our code just copy the python code provided below, and create a database with the name nycautocollision, and add your username and password to the connection.

How to run the script:

Just make sure to download the following libraries :

requests

pandas

mysql

sqlalchemy

Io

And create the database as mentioned in the previous section.

**Python Script:**

**# -\*- coding: utf-8 -\*-**

**"""**

**Created on Tue Oct 20 01:37:24 2020**

**@author: mohamed**

**"""**

**#import libraries**

**import requests**

**import pandas as pd**

**import mysql.connector**

**#import pymysql**

**from sqlalchemy import create\_engine;**

**import io**

**#1- extract data**

**#get the first 10 rows of the death causes dataset**

**url="https://data.cityofnewyork.us/resource/jb7j-dtam.csv"**

**r = requests.get(url)**

**rContent=r.content**

**pd.set\_option('display.max\_columns', None)**

**deathCausesdf = pd.read\_csv(io.StringIO(rContent.decode('utf-8')),nrows=10);**

**#print(deathCausesdf)**

**print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Car Collisions\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")**

**## insert data into time dimension**

**#extract time from the dataset**

**url="https://data.cityofnewyork.us/resource/h9gi-nx95.csv?$select=crash\_date,crash\_time"**

**r = requests.get(url)**

**rContent=r.content**

**pd.set\_option('display.max\_columns', None)**

**#add the extracted columns to a data frame.**

**timeDf = pd.read\_csv(io.StringIO(rContent.decode('utf-8')));**

**#Convert to datetime data type**

**timeDf['crash\_date']=pd.to\_datetime(timeDf['crash\_date'])**

**# Add a year column**

**timeDf['year'] = [d.year for d in timeDf['crash\_date']]**

**#Add a month column**

**timeDf['month'] = [d.month for d in timeDf['crash\_date']]**

**#add column for the day of the week.**

**timeDf['day\_number\_of\_week'] = [d.strftime("%w") for d in timeDf['crash\_date']]**

**# add a time id**

**timeDf['timeId']=timeDf['crash\_date'].astype(str)+" "+timeDf['crash\_time']**

**#drop the duplicated rows**

**timeDf=timeDf.drop\_duplicates(subset='timeId', keep="first")**

**#print(timeDf)**

**# here we create a connection to mysql db "nycautocollision"**

**engine = create\_engine("mysql+pymysql://{user}:{pw}@127.0.0.1/{db}".format(user="root", pw="root", db="nycautocollision"))**

**#this line of code create table in the databse with the name time dimension and columns in df , and insert the data from the df**

**timeDf.to\_sql('time\_dimension', con = engine, if\_exists = 'replace', chunksize = 1000);**

**## Loading data to Contributing Factor Dimension**

**# Extracting the columns we are intersted in**

**url="https://data.cityofnewyork.us/resource/h9gi-nx95.csv?$select=contributing\_factor\_vehicle\_1,contributing\_factor\_vehicle\_2,contributing\_factor\_vehicle\_3,contributing\_factor\_vehicle\_4,contributing\_factor\_vehicle\_5"**

**r = requests.get(url)**

**rContent=r.content**

**pd.set\_option('display.max\_columns', None)**

**#adding the data to a dtat frame**

**contributingFactorDF = pd.read\_csv(io.StringIO(rContent.decode('utf-8')));**

**frames=[contributingFactorDF['contributing\_factor\_vehicle\_1'],contributingFactorDF['contributing\_factor\_vehicle\_2'],contributingFactorDF['contributing\_factor\_vehicle\_3'],contributingFactorDF['contributing\_factor\_vehicle\_4'],contributingFactorDF['contributing\_factor\_vehicle\_5']]**

**contributingFactorDF=pd.concat(frames, names='contributing\_factor', ignore\_index=True)**

**contributingFactorDF=pd.DataFrame(contributingFactorDF,columns=['contributing\_factor'])**

**contributingFactorDF=contributingFactorDF.drop\_duplicates(subset='contributing\_factor')**

**contributingFactorDF['contributing\_factor\_id']=[(i+1000) for i in contributingFactorDF.index]**

**#print(contributingFactorDF.shape)**

**#contributingFactorDF=contributingFactorDF.set\_index(pd.Index(range(0,37)))**

**#print(contributingFactorDF)**

**#Loading the contributing factor dimension to the datawraehouse**

**contributingFactorDF.to\_sql('contributing\_factor', con = engine, if\_exists = 'replace', chunksize = 100, index=False);**

**#vehicle dimension**

**url="https://data.cityofnewyork.us/resource/bm4k-52h4.csv?$select=vehicle\_id,vehicle\_type, vehicle\_make,vehicle\_model,vehicle\_year"**

**r = requests.get(url)**

**rContent=r.content**

**pd.set\_option('display.max\_columns', None)**

**vehicle = pd.read\_csv(io.StringIO(rContent.decode('utf-8')));**

**vehicle=vehicle.drop\_duplicates();**

**#vehicle['vehicle\_id']=vehicle['vehicle\_type'].astype(str)+" "+vehicle['vehicle\_make']+vehicle['vehicle\_model']+vehicle['vehicle\_year'].astype(str)**

**vehicle.to\_sql('vehicle', con = engine, if\_exists = 'replace', chunksize = 1000, index=False);**

**#place Dimension**

**url="https://data.cityofnewyork.us/resource/h9gi-nx95.csv?$select=borough,zip\_code,latitude,longitude "**

**r = requests.get(url)**

**rContent=r.content**

**pd.set\_option('display.max\_columns', None)**

**place = pd.read\_csv(io.StringIO(rContent.decode('utf-8')));**

**place['zip\_code']=place['zip\_code'].fillna(1)**

**place['latitude']=place['latitude'].fillna(1)**

**place['longitude']=place['longitude'].fillna(1)**

**place['place\_id']=place['zip\_code']+place['latitude']+place['longitude']**

**place=place.drop\_duplicates()**

**#place=place.set\_index(pd.Index(range(0,37)))**

**#print(place.head(10))**

**place.to\_sql('place', con = engine, if\_exists = 'replace', chunksize = 100, index=False);**

**### in the next code we are going to extract all files using APIs and combine them in one dataframe through outer joins**

**pd.set\_option('display.max\_columns', None)**

**print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")**

**print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Collision Crashes\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")**

**#Extract collision dataset using Api(we passes the coulmn names parameters in the url select=..)**

**url="https://data.cityofnewyork.us/resource/h9gi-nx95.csv?$select=zip\_code,collision\_id,crash\_date,crash\_time,contributing\_factor\_vehicle\_1,contributing\_factor\_vehicle\_2,contributing\_factor\_vehicle\_3,borough,latitude,longitude,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"**

**r = requests.get(url)**

**rContent=r.content**

**#add to a python dataFrame**

**collisionCrashes = pd.read\_csv(io.StringIO(rContent.decode('utf-8')));**

**collisionCrashes['crash\_date']=pd.to\_datetime(collisionCrashes['crash\_date'])**

**collisionCrashes['year'] = [d.year for d in collisionCrashes['crash\_date']]**

**collisionCrashes['month'] = [d.month for d in collisionCrashes['crash\_date']]**

**collisionCrashes['day\_number\_of\_week'] = [d.strftime("%w") for d in collisionCrashes['crash\_date']]**

**collisionCrashes['timeId']=collisionCrashes['crash\_date'].astype(str)+" "+collisionCrashes['crash\_time']**

**collisionCrashes['zip\_code']=collisionCrashes['zip\_code'].fillna(1)**

**collisionCrashes['latitude']=collisionCrashes['latitude'].fillna(1)**

**collisionCrashes['longitude']=collisionCrashes['longitude'].fillna(1)**

**collisionCrashes['place\_id']=collisionCrashes['zip\_code']+collisionCrashes['latitude']+collisionCrashes['longitude']**

**print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")**

**print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Collision vehicles\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")**

**#Extract collision vehicles dataset using Api(we passes the coulmn names parameters in the url select=..)**

**url="https://data.cityofnewyork.us/resource/bm4k-52h4.csv?$select=vehicle\_id,collision\_id,vehicle\_type, vehicle\_make,vehicle\_model,vehicle\_year"**

**r = requests.get(url)**

**rContent=r.content**

**pd.set\_option('display.max\_columns', None)**

**vehicle = pd.read\_csv(io.StringIO(rContent.decode('utf-8')));**

**print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")**

**print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* race\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")**

**url="https://data.cityofnewyork.us/resource/kku6-nxdu.csv?$select=jurisdiction\_name,percent\_pacific\_islander,percent\_hispanic\_latino,percent\_american\_indian,percent\_asian\_non\_hispanic,percent\_white\_non\_hispanic,percent\_black\_non\_hispanic,percent\_other\_ethnicity"**

**r = requests.get(url)**

**rContent=r.content**

**pd.set\_option('display.max\_columns', None)**

**race = pd.read\_csv(io.StringIO(rContent.decode('utf-8')));**

**race['zip\_code']=race['jurisdiction\_name'];**

**crashes = pd.merge(collisionCrashes, vehicle, how='outer', on=['collision\_id', 'collision\_id'])**

**#We will join the contributing facto data frame wit =h the crashesdemographs data frame to get the contributing factor's surregate key**

**crashesdemographs=pd.merge(crashes, race, how='outer', on=['zip\_code'])**

**contributingFactorDF['contributing\_factor\_vehicle\_1']=contributingFactorDF['contributing\_factor']**

**crashesdemographs=pd.merge(crashesdemographs, contributingFactorDF, how='left', on=['contributing\_factor\_vehicle\_1'])**

**crashesdemographs['contributing\_factor1\_id']=crashesdemographs['contributing\_factor\_id']**

**crashesdemographs=crashesdemographs.drop(['contributing\_factor\_id','contributing\_factor'],axis=1)**

**contributingFactorDF['contributing\_factor\_vehicle\_2']=contributingFactorDF['contributing\_factor']**

**crashesdemographs=pd.merge(crashesdemographs, contributingFactorDF, how='left', on=['contributing\_factor\_vehicle\_2'])**

**crashesdemographs['contributing\_factor2\_id']=crashesdemographs['contributing\_factor\_id']**

**crashesdemographs=crashesdemographs.drop(['contributing\_factor\_id','contributing\_factor','contributing\_factor\_vehicle\_1\_y'],axis=1)**

**contributingFactorDF['contributing\_factor\_vehicle\_3']=contributingFactorDF['contributing\_factor']**

**crashesdemographs=pd.merge(crashesdemographs, contributingFactorDF, how='left', on=['contributing\_factor\_vehicle\_3'])**

**crashesdemographs['contributing\_factor3\_id']=crashesdemographs['contributing\_factor\_id']**

**print(crashesdemographs.shape)**

**crashesdemographs.drop\_duplicates()**

**print(crashesdemographs.shape)**

**CrashInjuryFactsDf=crashesdemographs[['vehicle\_id','timeId','contributing\_factor1\_id','contributing\_factor2\_id','contributing\_factor3\_id','place\_id','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']]**

**CrashInjuryFactsDf.to\_sql('crash\_injury\_fact', con = engine, if\_exists = 'replace', chunksize = 100, index=False);**

## 4. Screen shots and brief descriptions of the final schema:

-- --------------------------------------------------------

-- Host: 127.0.0.1

-- Server version: 10.5.4-MariaDB - mariadb.org binary distribution

-- Server OS: Win64

-- HeidiSQL Version: 11.0.0.5919

-- --------------------------------------------------------

/\*!40101 SET @OLD\_CHARACTER\_SET\_CLIENT=@@CHARACTER\_SET\_CLIENT \*/;

/\*!40101 SET NAMES utf8 \*/;

/\*!50503 SET NAMES utf8mb4 \*/;

/\*!40014 SET @OLD\_FOREIGN\_KEY\_CHECKS=@@FOREIGN\_KEY\_CHECKS, FOREIGN\_KEY\_CHECKS=0 \*/;

/\*!40101 SET @OLD\_SQL\_MODE=@@SQL\_MODE, SQL\_MODE='NO\_AUTO\_VALUE\_ON\_ZERO' \*/;

-- Dumping database structure for nycautocollision

CREATE DATABASE IF NOT EXISTS `nycautocollision` /\*!40100 DEFAULT CHARACTER SET latin1 \*/;

USE `nycautocollision`;

-- Dumping structure for table nycautocollision.contributing\_factor

CREATE TABLE IF NOT EXISTS `contributing\_factor` (

`contributing\_factor` text DEFAULT NULL,

`contributing\_factor\_id` bigint(20) DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

-- Dumping structure for table nycautocollision.crash\_injury\_fact

CREATE TABLE IF NOT EXISTS `crash\_injury\_fact` (

`vehicle\_id` text DEFAULT NULL FOREIGN KEY REFERENCES `vehicle` (`vehicle\_id` ),

`timeId` text DEFAULT NULL FOREIGN KEY REFERENCES `time\_dimension` (`timeId` ),

`contributing\_factor1\_id` bigint(20) DEFAULT NULL,

`contributing\_factor2\_id` bigint(20) DEFAULT NULL,

`contributing\_factor3\_id` bigint(20) DEFAULT NULL,

`place\_id` double DEFAULT NULL,

`number\_of\_persons\_injured` double DEFAULT NULL,

`number\_of\_persons\_killed` double DEFAULT NULL,

`number\_of\_pedestrians\_injured` double DEFAULT NULL,

`number\_of\_pedestrians\_killed` double DEFAULT NULL,

`number\_of\_cyclist\_injured` double DEFAULT NULL,

`number\_of\_cyclist\_killed` double DEFAULT NULL,

`number\_of\_motorist\_injured` double DEFAULT NULL,

`number\_of\_motorist\_killed` double DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

-- Dumping structure for table nycautocollision.place

CREATE TABLE IF NOT EXISTS `place` (

`borough` text DEFAULT NULL,

`zip\_code` double DEFAULT NULL,

`latitude` double DEFAULT NULL,

`longitude` double DEFAULT NULL,

`place\_id` int NOT NULL PRIMARY KEY

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

-- Dumping structure for table nycautocollision.time\_dimension

CREATE TABLE IF NOT EXISTS `time\_dimension` (

`index` bigint(20) DEFAULT NULL,

`crash\_date` datetime DEFAULT NULL,

`crash\_time` text DEFAULT NULL,

`year` bigint(20) DEFAULT NULL,

`month` bigint(20) DEFAULT NULL,

`day\_number\_of\_week` text DEFAULT NULL,

`timeId` text NOT NULL PRIMARY KEY,

KEY `ix\_time\_dimension\_index` (`index`)

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

-- Dumping structure for table nycautocollision.vehicle

CREATE TABLE IF NOT EXISTS `vehicle` (

`vehicle\_id` text DEFAULT NOT NULL PRIMARY KEY

`vehicle\_type` text DEFAULT NULL,

`vehicle\_make` text DEFAULT NULL,

`vehicle\_model` text DEFAULT NULL,

`vehicle\_year` double DEFAULT NULL

) ENGINE=InnoDB DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

## 5.Screen shots and descriptions of the analytics (at least 3) on the dashboard applications developed based on the data warehouse data.:

We used the following 3 queries to export the data to excel and build/ update the charts:

Most common factors for crashes:

SELECT f.contributing\_factor, c.contributing\_factor1\_id, c.contributing\_factor2\_id, c.contributing\_factor3\_id, COUNT(contributing\_factor1\_id) AS facto1\_count

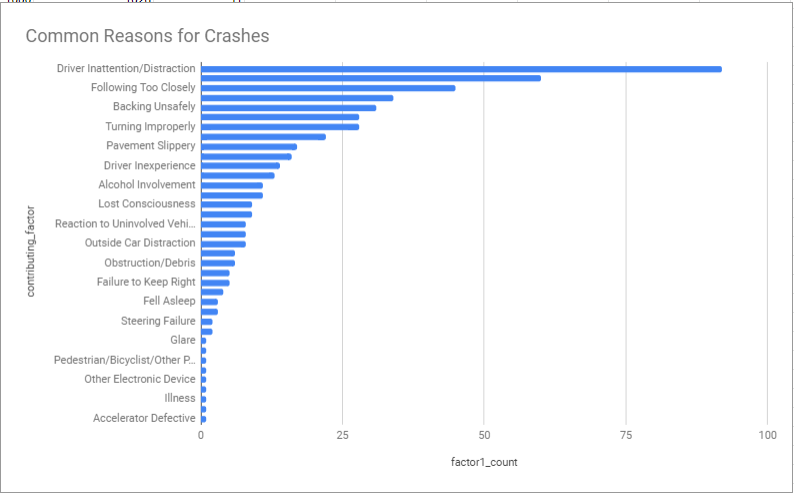
FROM crash\_injury\_fact c INNER JOIN contributing\_factor f

ON c.contributing\_factor1\_id=f.contributing\_factor\_id

GROUP BY contributing\_factor1\_id

ORDER BY facto1\_count;

Most common factors for crashes:



Most common factors for injuries or death:

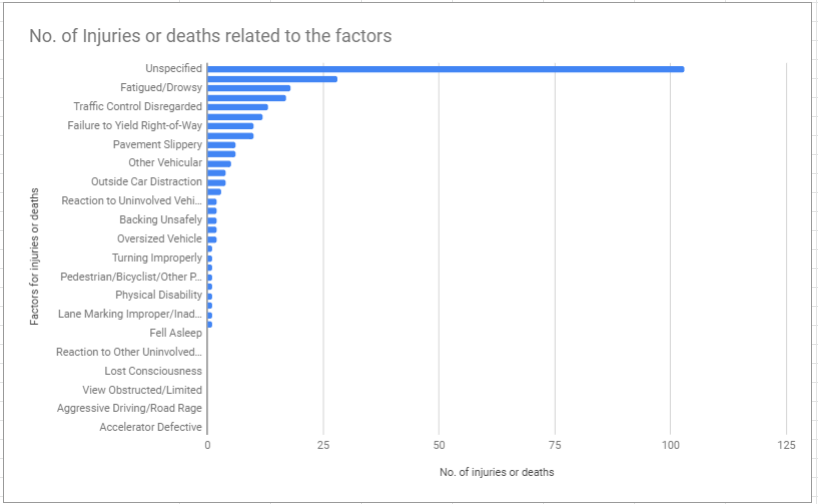
SELECT f.contributing\_factor, c.contributing\_factor1\_id, c.contributing\_factor2\_id, c.contributing\_factor3\_id, (sum(number\_of\_persons\_injured)+SUM(number\_of\_persons\_killed)) AS injuredorKilled

FROM crash\_injury\_fact c INNER JOIN contributing\_factor f

ON c.contributing\_factor1\_id=f.contributing\_factor\_id

GROUP BY contributing\_factor1\_id

ORDER BY injuredorKilled DESC;



Number of crashes per zipcode:

SELECT distinct p.zip\_code, COUNT(p.zip\_code) AS crashes

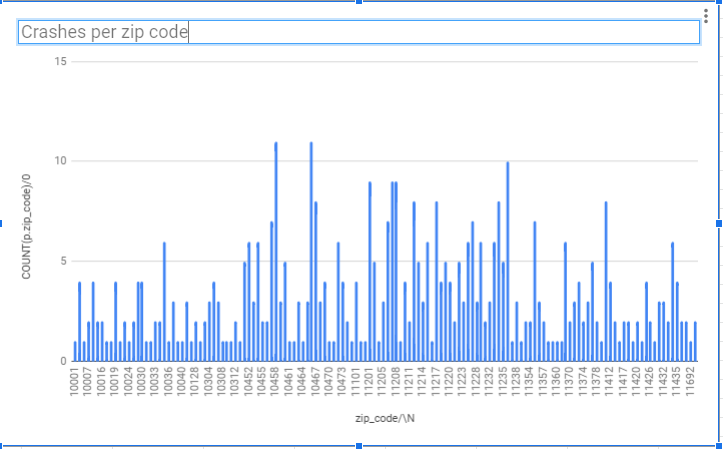
FROM

crash\_injury\_fact c left JOIN place p

ON p.place\_id=c.place\_id

GROUP BY p.zip\_code

ORDER BY crashes desc;



## 6- My personal Experience:

I personally did the DImensional model, the architectural diagram, and The ETL part. It took longer than I expected. I am still not 100% sure about how clean is the data, and how authentic is our analysis.I also created some sql queries that helped with the dashboards.

Figuring out how to deal with the surrogate keys was a big challenge for mr.

If I were to do the project again, I would have started earlier. I would have better led and managed my team, and helped them do their best. Defined the team structures and deadlines upfront. Asked for more help.

We plain to publish another version to fix bugs and update the current version.