# Detailed Report on Smart City Traffic & Accident Analytics

---- Sayan Das

## Note:

There were two files available—*road\_traffic\_sensor\_data.csv* and *traffic\_sensor\_data.csv*. Both contained sensor information and traffic conditions. However, to generate this report, I only needed one of these datasets. I chose *road\_traffic\_sensor\_data.csv* because the other file (*traffic\_sensor\_data.csv*) contained questionable data like pollution sensor, giving speed data..

## 1. Data Loading & Pre-Processing

I began by loading the provided CSV files for sensor and accident data into Python using the Pandas library. This step involved standardizing column names (converted to lower-case) for consistency.

**Python Code Snippet:**

import pandas as pd  
  
# Load CSV files  
df\_sensor = pd.read\_csv('road\_traffic\_sensor\_data.csv')  
df\_accident = pd.read\_csv('traffic\_accident\_data.csv')  
  
# Standardize column names to lower-case  
df\_sensor.columns = [col.lower() for col in df\_sensor.columns]  
df\_accident.columns = [col.lower() for col in df\_accident.columns]

This allowed me to work with a consistent dataset before moving on to the quality and transformation steps.

## 2. Data Quality Checks & Cleaning

To ensure reliable analytics, I performed several data quality checks: - **Uniqueness Check:** Verified that the primary key columns (sensor\_id in sensor data and accident\_id in accident data) are unique. - **Missing Data Check:** Assessed the presence of null values. I opted to drop any rows with missing data for simplicity.

**Python Code Snippet:**

# Check for primary key uniqueness  
if df\_sensor['sensor\_id'].nunique() != len(df\_sensor):  
 print("Warning: Duplicate sensor\_id values found in sensor data!")  
else:  
 print("All sensor\_id values are unique in sensor data.")  
  
if df\_accident['accident\_id'].nunique() != len(df\_accident):  
 print("Warning: Duplicate accident\_id values found in accident data!")  
else:  
 print("All accident\_id values are unique in accident data.")  
  
# Check for missing values  
print("Missing values in sensor data:")  
print(df\_sensor.isnull().sum())  
  
print("\nMissing values in accident data:")  
print(df\_accident.isnull().sum())  
  
# Drop rows with missing data  
df\_sensor.dropna(inplace=True)  
df\_accident.dropna(inplace=True)

Additionally, I converted the date\_time columns into datetime objects to facilitate further time-based transformations.

df\_sensor['date\_time'] = pd.to\_datetime(df\_sensor['date\_time'])

df\_accident['date\_time'] = pd.to\_datetime(df\_accident['date\_time'])

## 3. Data Transformation: Creating Dimension and Fact Tables

Following data cleaning, I transformed the raw data into a structured star schema by creating dimension tables and fact tables.

### 3.1 Creating Dimension Tables

I built the following dimension tables:

* **dim\_time:**  
  I combined unique date\_time values from both datasets, extracted additional attributes (year, month, day, hour, day\_of\_week), and created a surrogate key time\_id.
* **Python Code Snippet:**
* all\_times = pd.concat([df\_sensor[['date\_time']], df\_accident[['date\_time']]])  
  all\_times = all\_times.drop\_duplicates().reset\_index(drop=True)  
    
  all\_times['year'] = all\_times['date\_time'].dt.year  
  all\_times['month'] = all\_times['date\_time'].dt.month  
  all\_times['day'] = all\_times['date\_time'].dt.day  
  all\_times['hour'] = all\_times['date\_time'].dt.hour  
  all\_times['day\_of\_week'] = all\_times['date\_time'].dt.dayofweek  
    
  all\_times.reset\_index(inplace=True)  
  all\_times.rename(columns={'index': 'time\_id'}, inplace=True)  
  all\_times['time\_id'] = all\_times['time\_id'] + 1  
    
  dim\_time = all\_times[['time\_id', 'date\_time', 'year', 'month', 'day', 'hour', 'day\_of\_week']]
* **dim\_location:**  
  I consolidated unique locations from both datasets and assigned a surrogate key location\_id.
* **Python Code Snippet:**
* locations\_sensor = df\_sensor[['location']].drop\_duplicates()  
  locations\_accident = df\_accident[['location']].drop\_duplicates()  
  all\_locations = pd.concat([locations\_sensor, locations\_accident]).drop\_duplicates().reset\_index(drop=True)  
    
  all\_locations.reset\_index(inplace=True)  
  all\_locations.rename(columns={'index': 'location\_id'}, inplace=True)  
  all\_locations['location\_id'] = all\_locations['location\_id'] + 1  
    
  dim\_location = all\_locations[['location\_id', 'location']]
* **Additional Dimensions (for Accident Data):**  
  I created separate dimensions for vehicle\_type, weather\_condition, and road\_condition with their own surrogate keys (vehicle\_id, weather\_id, and road\_id respectively).
* **Python Code Snippet:**
* # Vehicle Dimension  
  dim\_vehicle = df\_accident[['vehicle\_type']].drop\_duplicates().reset\_index(drop=True)  
  dim\_vehicle.reset\_index(inplace=True)  
  dim\_vehicle.rename(columns={'index': 'vehicle\_id'}, inplace=True)  
  dim\_vehicle['vehicle\_id'] = dim\_vehicle['vehicle\_id'] + 1  
  dim\_vehicle = dim\_vehicle[['vehicle\_id', 'vehicle\_type']]  
    
  # Weather Dimension  
  dim\_weather = df\_accident[['weather\_condition']].drop\_duplicates().reset\_index(drop=True)  
  dim\_weather.reset\_index(inplace=True)  
  dim\_weather.rename(columns={'index': 'weather\_id'}, inplace=True)  
  dim\_weather['weather\_id'] = dim\_weather['weather\_id'] + 1  
  dim\_weather = dim\_weather[['weather\_id', 'weather\_condition']]  
    
  # Road Condition Dimension  
  dim\_road = df\_accident[['road\_condition']].drop\_duplicates().reset\_index(drop=True)  
  dim\_road.reset\_index(inplace=True)  
  dim\_road.rename(columns={'index': 'road\_id'}, inplace=True)  
  dim\_road['road\_id'] = dim\_road['road\_id'] + 1  
  dim\_road = dim\_road[['road\_id', 'road\_condition']]

### 3.2 Creating Fact Tables

Next, I built the fact tables by merging the cleaned data with the respective dimensions: - **fact\_traffic:**  
This fact table includes the key measures from sensor data along with foreign keys linking to dim\_time and dim\_location.

**Python Code Snippet:** ```python # Merge with dim\_time and rename to fk\_time\_id fact\_traffic = df\_sensor.merge(dim\_time[[‘time\_id’, ‘date\_time’]], on=‘date\_time’, how=‘left’) fact\_traffic.rename(columns={‘time\_id’: ‘fk\_time\_id’}, inplace=True)

# Merge with dim\_location and rename to fk\_location\_id fact\_traffic = fact\_traffic.merge(dim\_location, on=‘location’, how=‘left’) fact\_traffic.rename(columns={‘location\_id’: ‘fk\_location\_id’}, inplace=True)

# Select only the necessary columns fact\_traffic = fact\_traffic[[‘sensor\_id’, ‘fk\_time\_id’, ‘fk\_location\_id’, ‘vehicle\_count’, ‘average\_speed’, ‘congestion\_level’]] ```

* **fact\_accident:**  
  This fact table holds accident-specific measures and references dimensions including dim\_time, dim\_location, dim\_vehicle, dim\_weather, and dim\_road.
* **Python Code Snippet:**
* # Merge with dim\_time and rename to fk\_time\_id  
  fact\_accident = df\_accident.merge(dim\_time[['time\_id', 'date\_time']], on='date\_time', how='left')  
  fact\_accident.rename(columns={'time\_id': 'fk\_time\_id'}, inplace=True)  
    
  # Merge with dim\_location and rename to fk\_location\_id  
  fact\_accident = fact\_accident.merge(dim\_location, on='location', how='left')  
  fact\_accident.rename(columns={'location\_id': 'fk\_location\_id'}, inplace=True)  
    
  # Merge with additional dimensions  
  fact\_accident = fact\_accident.merge(dim\_vehicle.rename(columns={'vehicle\_id': 'fk\_vehicle\_id'}),  
   on='vehicle\_type', how='left')  
  fact\_accident = fact\_accident.merge(dim\_weather.rename(columns={'weather\_id': 'fk\_weather\_id'}),  
   on='weather\_condition', how='left')  
  fact\_accident = fact\_accident.merge(dim\_road.rename(columns={'road\_id': 'fk\_road\_id'}),  
   on='road\_condition', how='left')  
    
  # Select only the necessary columns  
  fact\_accident = fact\_accident[['accident\_id', 'fk\_time\_id', 'fk\_location\_id', 'fk\_vehicle\_id', 'fk\_weather\_id', 'fk\_road\_id','accident\_severity', 'number\_of\_vehicles', 'casualties','traffic\_density']]

## 4. Loading Transformed Data to MySQL

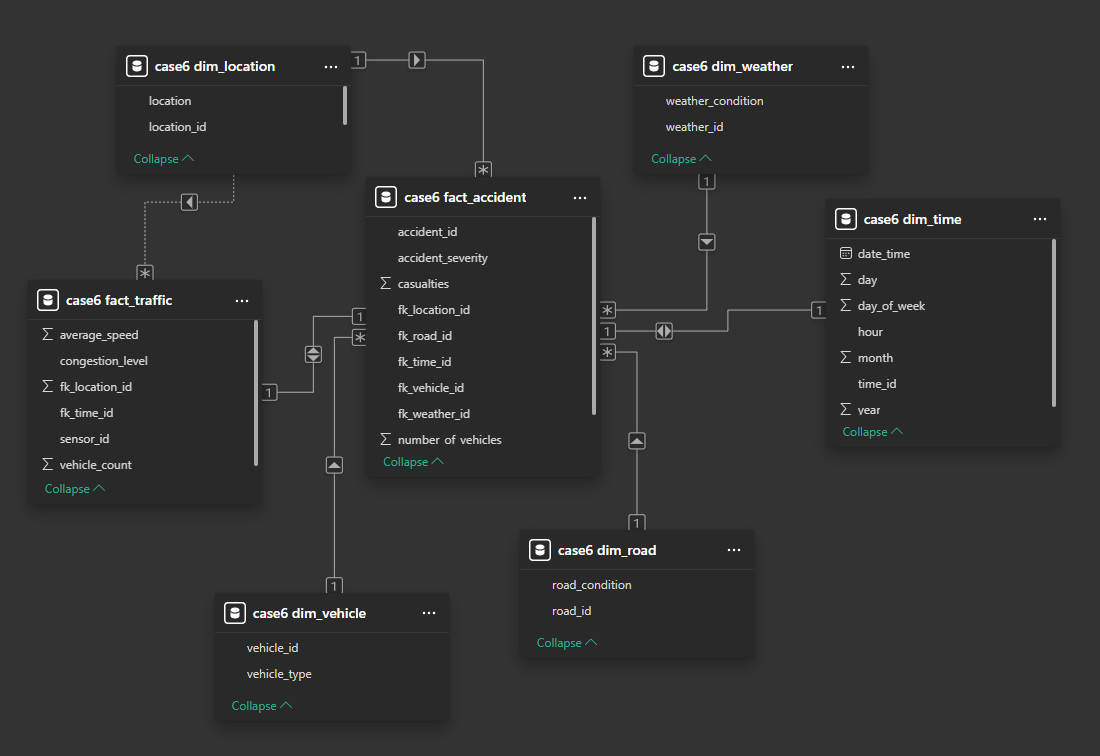
After the ETL process, I loaded the dimension and fact tables into a MySQL database using SQLAlchemy. I configured the connection parameters and used the to\_sql() method with if\_exists='replace' to update the tables.

**Python Code Snippet:**

from sqlalchemy import create\_engine  
  
# MySQL connection parameters  
username = 'root'  
password = '12345'  
host = 'localhost'  
port = '3306'  
database = 'case6'  
  
# Create the SQLAlchemy engine  
engine = create\_engine(f'mysql+pymysql://{username}:{password}@{host}:{port}/{database}')  
  
# Load tables into MySQL  
dim\_time.to\_sql('dim\_time', con=engine, index=False, if\_exists='replace')  
dim\_location.to\_sql('dim\_location', con=engine, index=False, if\_exists='replace')  
dim\_vehicle.to\_sql('dim\_vehicle', con=engine, index=False, if\_exists='replace')  
dim\_weather.to\_sql('dim\_weather', con=engine, index=False, if\_exists='replace')  
dim\_road.to\_sql('dim\_road', con=engine, index=False, if\_exists='replace')  
fact\_traffic.to\_sql('fact\_traffic', con=engine, index=False, if\_exists='replace')  
fact\_accident.to\_sql('fact\_accident', con=engine, index=False, if\_exists='replace')  
  
print("\nAll tables have been loaded successfully into MySQL!")

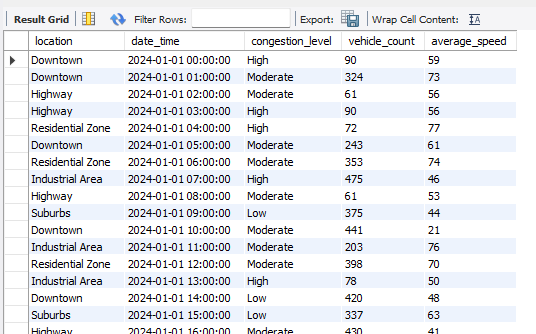
## 5. SQL Queries for Insights

Schema:



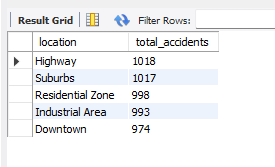
* **Traffic Congestion (Real time data not available)**

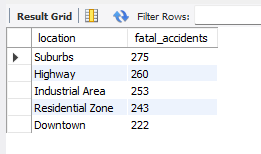
USE case6;  
  
-- 1. Show traffic congestion by location and time  
SELECT  
 dl.location,  
 dt.date\_time,  
 ft.congestion\_level,  
 ft.vehicle\_count,  
 ft.average\_speed  
FROM fact\_traffic AS ft  
JOIN dim\_time AS dt   
 ON ft.fk\_time\_id = dt.time\_id  
JOIN dim\_location AS dl  
 ON ft.fk\_location\_id = dl.location\_id  
ORDER BY dt.date\_time;



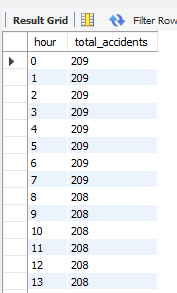
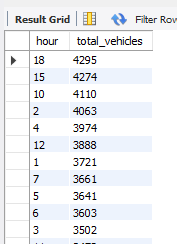
* **Accident-Prone Areas**

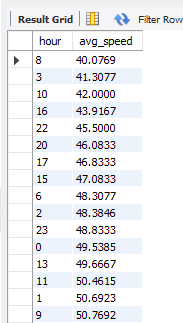
-- 2. Show top 10 accident-prone areas  
SELECT  
 dl.location,  
 COUNT(\*) AS total\_accidents  
FROM fact\_accident AS fa  
JOIN dim\_location AS dl  
 ON fa.fk\_location\_id = dl.location\_id  
GROUP BY dl.location  
ORDER BY total\_accidents DESC  
LIMIT 10;

  
  
-- Top locations with the highest count of 'Fatal' accidents  
SELECT   
 dl.location,  
 COUNT(\*) AS fatal\_accidents  
FROM fact\_accident AS fa  
JOIN dim\_location AS dl  
 ON fa.fk\_location\_id = dl.location\_id  
WHERE fa.accident\_severity = 'Fatal'  
GROUP BY dl.location  
ORDER BY fatal\_accidents DESC  
LIMIT 10;



* **Peak Hour Analysis**

-- 3. Peak hour analysis for accidents  
SELECT  
 dt.hour,  
 COUNT(\*) AS total\_accidents  
FROM fact\_accident AS fa  
JOIN dim\_time AS dt  
 ON fa.fk\_time\_id = dt.time\_id  
GROUP BY dt.hour  
ORDER BY total\_accidents DESC;  
  
-- 4. Peak Hour for Traffic Volume  
SELECT  
 dt.hour,  
 SUM(ft.vehicle\_count) AS total\_vehicles  
FROM fact\_traffic AS ft  
JOIN dim\_time AS dt  
 ON ft.fk\_time\_id = dt.time\_id  
GROUP BY dt.hour  
ORDER BY total\_vehicles DESC;  
  
-- 5. Average speed by hour  
SELECT  
 dt.hour,  
 AVG(ft.average\_speed) AS avg\_speed  
FROM fact\_traffic AS ft  
JOIN dim\_time AS dt  
 ON ft.fk\_time\_id = dt.time\_id  
GROUP BY dt.hour  
ORDER BY avg\_speed;



-- 4. Peak hour analysis for traffic volume

SELECT

dt.hour,

SUM(ft.vehicle\_count) AS total\_vehicles

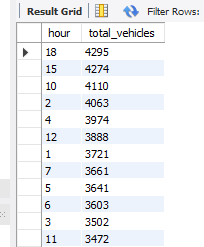
FROM fact\_traffic AS ft

JOIN dim\_time AS dt

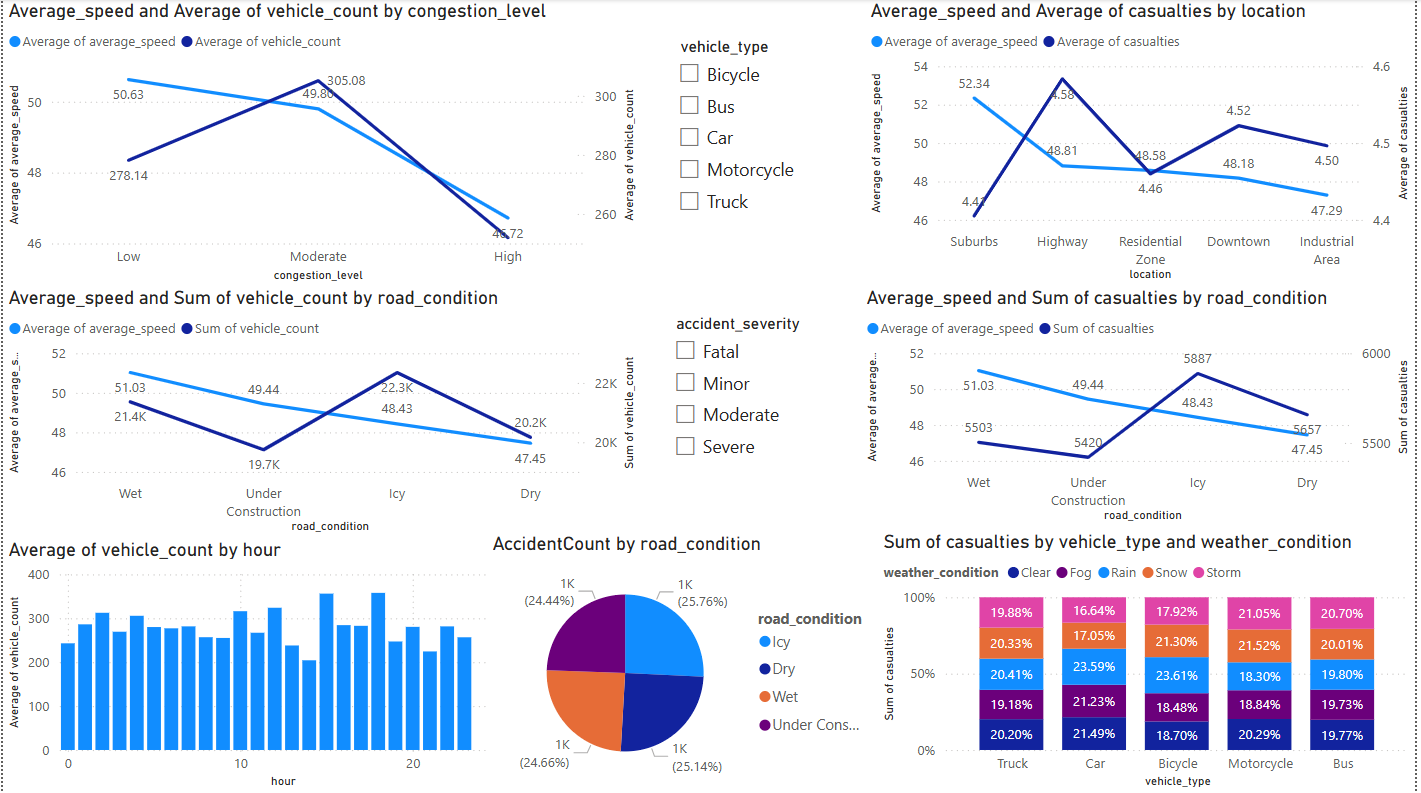
ON ft.fk\_time\_id = dt.time\_id

GROUP BY dt.hour

ORDER BY total\_vehicles DESC;



## 6. PowerBI Report



# Conclusion

* **Focus on High‐Congestion Zones/Times:** Implement dynamic traffic signals, ramp metering, or congestion pricing in the hours and locations where congestion is severe and speed is low.
* **Improve Road Safety in Hazardous Conditions:**
  + **Icy or Wet Roads:** Deploy salting trucks, improved drainage, or better signage.
  + **Under Construction Areas:** Enhance signage, set lower speed limits, and ensure construction zones are well-lit.
* **Target Vehicle-Specific Interventions:**
  + **Buses**: Possibly large casualties due to high passenger count—enforce rigorous driver training and vehicle maintenance.
  + **Motorcycles**: High vulnerability in bad weather—promote protective gear, stricter licensing.