ECE 356 Project

Dataset: UK Car Accidents 2005-2015

Team-38

Members: Eric Bian, Lovish Chugh, Jun Zheng

Gitlab: https://git.uwaterloo.ca/ece356-f2023/Team-38

Table of Contents

Raw CSV Attributes

Accidents
Accident_Index, Location_Easting_OSGR, Location_Northing_OSGR, Longitude, Latitude, Police_Force, Accident_Severity, Number_of_Vehicles, Number_of_Casualties, Date, Day_of_Week, Time, Local_Authority_(District), Local_Authority_(Highway), 1st_Road_Class, 1st_Road_Number, Road_Type, Speed_limit, Junction_Detail, Junction_Control, 2nd_Road_Class, 2nd_Road_Number, Pedestrian_Crossing-Human_Control, Pedestrian_Crossing-Physical_Facilities, Light_Conditions, Weather_Conditions, Road_Surface_Conditions, Special_Conditions_at_Site, Carriageway_Hazards, Urban_or_Rural_Area, Did_Police_Officer_Attend_Scene_of_Accident, LSOA_of_Accident_Location

Casualties
Accident_Index, Vehicle_Reference, Casualty_Reference, Casualty_Class, Sex_of_Casualty, Age_of_Casualty, Age_Band_of_Casualty, Casualty_Severity, Pedestrian_Location, Pedestrian_Movement, Car_Passenger, Bus_or_Coach_Passenger, Pedestrian_Road_Maintenance_Worker, Casualty_Type, Casualty_Home_Area_Type

Vehicles
Accident_Index, Vehicle_Reference, Vehicle_Type, Towing_and_Articulation, Vehicle_Manoeuvre, Vehicle_Location-Restricted_Lane, Junction_Location, Skidding_and_Overturning, Hit_Object_in_Carriageway, Vehicle_Leaving_Carriageway, Hit_Object_off_Carriageway, 1st_Point_of_Impact, Was_Vehicle_Left_Hand_Drive?, Journey_Purpose_of_Driver, Sex_of_Driver, Age_of_Driver, Age_Band_of_Driver, Engine_Capacity_(CC), Propulsion_Code, Age_of_Vehicle, Driver_IMD_Decile, Driver_Home_Area_Type

Client Application

This system is structured as a Command-Line Interface (CLI). This setup allows for a logical and straightforward user experience, enhancing ease of use. Through the CLI, users are smoothly guided across a range of features and choices, facilitating effective and streamlined operation. Such an arrangement proves particularly advantageous for the orderly management and querying of complex data sets.

To run the CLI, run `python3 client.py` and follow the prompts in the terminal, and provide the input when the CLI prompts it.

Main function

`main:`

1. This function acts as the central hub of the application, orchestrating the flow of actions based on user input.
2. It calls `connect_to_database` to establish a database connection.
3. Once connected, it enters a loop where it presents a menu to the user with different options, each corresponding to various database operations – like querying accident details or updating driver information.
4. The user's choice determines which function is called. For instance, choosing "1" would execute accidents_road_conditions, which performs a specific query on the database.
5. For update operations, the function collects additional inputs from the user, such as the accident index or new data to update, and passes these to the respective update functions.
6. After executing an option, the loop presents the menu again until the user decides to exit by entering "-1". At this point, the database connection and cursor are closed to end the session.

Connection function

`connection_to_database:`

- The function connects to the MySQL database
- The credentials are set in the function and connects to db356_team38
- Output the success message when successfully connect to database

Get functions

`list_accidents_in_area`:

- **Functionality:** This function fetches the first 100 accidents from a specified urban or rural area, helping to pinpoint accident hotspots in that specific zone.
- **Database Interaction**: It executes a SELECT query on the Location table, filtering results based on the 'Urban_or_Rural_Area' parameter. The query retrieves details like accident index, geographical coordinates, and junction details.
- **Practical Use**: Useful for traffic management authorities to identify areas needing safety improvements or for research into urban vs. rural accident trends.

`accidents_weather_conditions`:

- **Functionality**: This function lists the total number of accidents under different weather conditions, offering insights into how weather impacts road safety.
- **Database Interaction**: It involves a JOIN operation between the Conditions and Accidents tables, grouping results by weather conditions and counting accidents.
- **Practical Use**: Vital for meteorological studies on road safety and for traffic authorities to strategize during adverse weather conditions.

`accidents_speed_limits`:

- **Functionality**: This function displays the count of accidents occurring at various speed limits, highlighting how speed impacts road accidents.
- **Database Interaction**: It conducts a JOIN between the Road and Accidents tables and uses GROUP BY on 'Speed_limit' to aggregate accident data.
- **Practical Use**: Essential for road safety analysis, helping in the formulation of speed regulation policies.

`pedestrian_involved_accidents`:

- **Functionality:** This function focuses on accidents involving pedestrians, providing specific details like human control and physical facilities at the accident location.
- **Database Interaction**: It executes a complex SQL query joining the Casualties, Accidents, and Location tables, filtering for pedestrian-related accidents.
- **Practical Use**: Key for urban planning and pedestrian safety initiatives, highlighting dangerous zones for pedestrians.

`driver_demographics`:

- **Functionality:** This function analyses accidents based on driver demographics like sex and age band, aiding in demographic risk assessment.
- **Database Interaction:** It joins the Driver, Vehicles, and Accidents tables, grouping results by sex and age band of drivers, and counting distinct accidents.
- **Practical Use:** Crucial for insurance companies in assessing risk profiles and for driving safety programs targeting specific demographic groups.

`accidents_road_conditions`:

- **Functionality**: This function categorizes accidents based on road surface conditions, elucidating how these conditions affect accident rates.
- **Database Interaction**: It performs a JOIN operation between the Conditions and Accidents tables and groups results by road surface conditions.
- **Practical Use**: Important for road maintenance authorities to prioritize repairs and for safety awareness during specific road conditions.

`accidents_light_conditions`:

- **Functionality:** This function provides data on accidents under various lighting conditions, indicating the influence of visibility on road safety.
- **Database Interaction:** It queries the Conditions table, grouping accidents by light conditions and counting them.
- **Practical Use:** Useful for urban lighting planning and for driving safety advisories during different times of the day.

`accidents_at_intersections`:

- **Functionality:** This function reveals the number of accidents occurring at intersections controlled by either stop signs or auto traffic signals.
- **Database Interaction**: It filters the Location table for accidents at specific types of junction controls and counts them.
- **Practical Use:** Vital for traffic management and infrastructure planning, focusing on intersection safety improvements.

`accidents_involving_young_drivers`:

- **Functionality**: This function identifies the total number of accidents involving young drivers, spotlighting this high-risk group.
- **Database Interaction**: It queries the Driver table, filtering for specific age bands indicative of young drivers, and counts distinct accidents.
- **Practical Use:** Crucial for targeted road safety campaigns and educational programs aimed at young drivers.

`accidents_by_time_of_day`:

- **Functionality:** This function segments accidents based on the time of day, offering insights into when accidents are most likely to occur.
- **Database Interaction:** It groups accidents from the Timestamp table into time slots (morning, afternoon, night) and counts them.
- **Practical Use:** Useful for traffic flow management and for understanding the impact of daily traffic patterns on accident rates.

`top_accidents_with_most_casualties`:

- **Functionality**: This function lists the top 10 accidents with the highest number of casualties, providing detailed information about each.
- **Database Interaction**: It performs a LEFT JOIN across the Accidents, Timestamp, Location, and Conditions tables, sorting by the number of casualties.
- **Practical Use**: Essential for emergency response planning and for identifying particularly dangerous locations or conditions.

`get_accident_info`:

- **Functionality**: This function retrieves detailed information about a specific accident, using its index as a reference.
- **Database Interaction**: It aggregates data from the Accidents, Timestamp, Location, and Conditions tables based on the given accident index.
- **Practical Use:** Useful for in-depth accident analysis, legal investigations, or insurance assessments.

Update functions

`update_driver_info`:

- **Functionality:** This function updates the driver's information associated with a specific accident, such as age, sex, and home area type.
- **Database Operation**: It performs an UPDATE operation on the Driver table, modifying the Age_of_Driver, Sex_of_Driver, and Driver_Home_Area_Type for a particular accident and vehicle reference.
- **Practical Use**: Essential for maintaining accurate and up-to-date driver information in accident records, particularly useful in ongoing investigations or insurance claims.

`update_accident_location`:

- **Functionality**: This function updates the geographical coordinates of an accident's location.
- **Database Operation**: It updates the Longitude and Latitude fields in the Location table for a specified accident index.
- **Practical Use**: Crucial for correcting or refining location data of accidents, aiding in accurate mapping and analysis of accident hotspots.

`update_vehicle_details`:

- **Functionality**: This function modifies details about a vehicle involved in an accident, including the vehicle type and its age.
- **Database Operation**: It executes an UPDATE command on the Vehicles table, altering the Vehicle_Type and Age_of_Vehicle for a given accident and vehicle reference.
- **Practical Use**: Vital for keeping vehicle-related data current, which is significant for statistical analyses of vehicle types involved in accidents and their conditions.

`update_accident_severity`:

- **Functionality**: This function allows for the modification of the recorded severity level of an accident.
- **Database Operation**: It updates the Accident_Severity field in the Accidents table for a particular accident index.
- **Practical Use**: Important for adjusting records to reflect post-investigation findings or court rulings, ensuring the accuracy of the accident severity classification.

`update_police_attendance`:

- **Functionality**: This function updates the record of whether police attended the scene of an accident.
- **Database Operation**: It modifies the Did_Police_Officer_Attend_Scene_of_Accident field in the Administration table for a specific accident.
- **Practical Use**: Essential for accurate record-keeping on police response, which can influence legal proceedings and insurance claims related to the accident.

Error handling

Each function includes try-except blocks to handle exceptions and errors gracefully, so the application doesn't crash unexpectedly and provides feedback to the user on what went wrong. For functions performing updates or insertions into the database, a failure partway through can leave the database in an inconsistent state. The use of connection.rollback() in the except block reverses any changes made during the transaction, safeguarding the data integrity.

Looped interface for continuous interaction

The main function uses a loop that keeps the application running, allowing the user to perform multiple operations in a single session. This loop continues until the user decides to exit, improving the application's usability.

Testing functions

TestListAccidentsInArea:

- Purpose: Tests the `list_accidents_in_area` function.
- setUp Method: Initializes a mock cursor before each test, ensuring a clean testing environment.
- `test_list_accidents_no_results`: Validates the function's behavior when no results are found in the specified area. It expects a specific print statement indicating no accidents were found.
- `test_list_accidents_exception`: Tests the function's response to an exception (e.g., database error). The function should catch the exception and print an error message.

TestAccidentsWeatherConditions

- Purpose: Tests the `accidents_weather_conditions` function.
- setUp Method: Sets up a mock cursor and sample weather condition data for testing.
- `test_accidents_weather_conditions_results`: Checks if the function correctly processes and prints the weather conditions and accident counts when data is available.
- `test_accidents_weather_conditions_no_results`: Ensures the function correctly handles scenarios where no data is available.
- `test_accidents_weather_conditions_exception`: Assesses the function's error handling by simulating a database error.

TestAccidentConditionQueries

- Purpose: Tests functions related to different accident conditions (road and light conditions).
- setUp Method: Prepares mock cursor and sample data for road and light conditions.
- `test_accidents_road_conditions_results`: Verifies that accidents_road_conditions correctly interprets and prints road condition data.
- `test_accidents_light_conditions_results`: Checks that `accidents_light_conditions` accurately processes and outputs data related to light conditions.

TestUpdateDriverInfo

- Purpose: Tests the `update_driver_info` function.
- setUp Method: Prepares mock cursor, mock connection, and sample data for a driver update scenario.
- `test_update_driver_info`: Validates the correct execution of an SQL update query and commits the transaction. Ensures that the success message is printed.
- `test_update_driver_info_exception`: Tests the function's behavior in case of an exception, such as a database error. It should handle the exception, print an error message, and roll back any changes.

General Characteristics

Mocking: Uses Python's unittest.mock library to simulate database cursor behavior and isolate tests from actual database interactions.

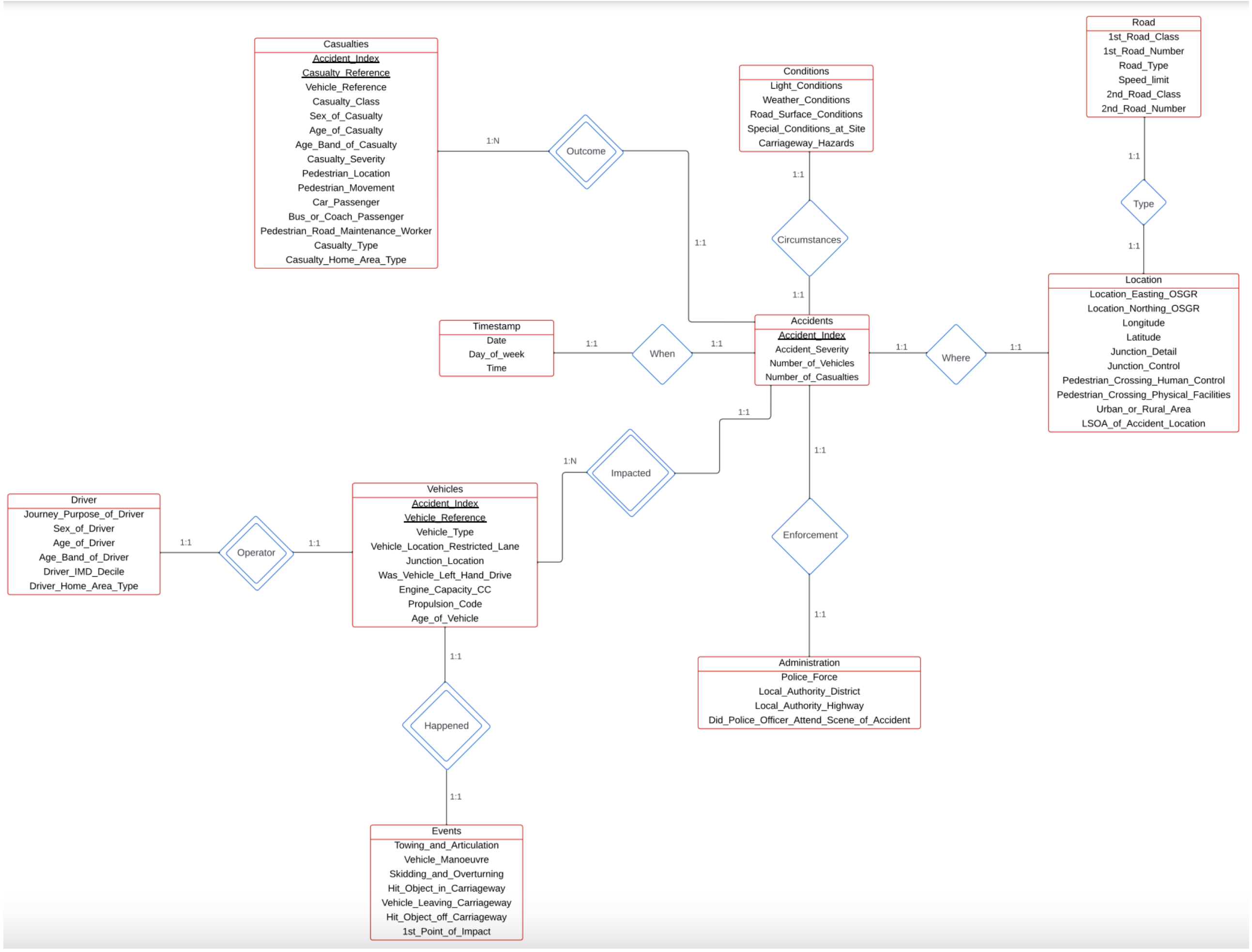
Error Handling Testing: Includes scenarios where exceptions are raised to ensure the functions handle errors gracefully.

Print Statement Testing: Uses patch from unittest.mock to intercept and test print statements for accurate user feedback.

Database Interaction Simulation: By mocking cursor and connection objects, the tests simulate database interactions without requiring an actual database connection.

Entity-Relationship Design

Diagram



Design Choices

The Entity-Relationship (ER) model is based on the original dataset comprising three main tables: Accidents0515.csv, Casualties0515.csv, and Vehicles0515.csv. The goal was to organize the data into a relational database structure, optimizing for logical groupings and efficient data retrieval.

Accidents Dataset

Accidents entity set

The Accidents entity set was chosen as the primary focus, with the following key attributes:

- Accident_Index: A unique identifier for each accident (Primary Key).
- Accident_Severity: Represents the severity level of the accident.
- Number_of_Vehicles: Indicates the count of vehicles involved in the accident.
- Number_of_Casualties: Represents the count of casualties in the accident.

Accidents Sub-classifications

A subset of attributes from the Accidents dataset was removed and split into 5 new entity sets:

- **Timestamp entity set**: Attributes related to the time of the accident.

- **Administration entity set**: Includes administrative enforcement-related details about the accident.
- **Location entity set**: Holds attributes related to the location of the accident.
    - **Road entity set**: A sub-classification entity set under Location, housing attributes specific to the roads where the accident occurred.
- **Condition entity set**: Represents attributes related to the conditions during the accident.

Casualties Dataset

Casualty entity set

The Casualties entity set includes the following attributes:

- Accident_Index: A foreign key linking to the Accidents entity set.
- Casualty_Reference: A unique identifier for each casualty (Primary Key).
- Vehicle_Reference: Identifies the vehicle associated with the casualty.
- Casualty_Class: Specifies the class of the casualty (e.g., driver, passenger, pedestrian).
- Sex_of_Casualty: Indicates the gender of the casualty.
- Age_of_Casualty: Represents the exact age of the casualty.
- Age_Band_of_Casualty: Categorizes the casualty's age into bands.
- Casualty_Severity: Describes the severity of the casualty's injuries.
- Pedestrian_Location: Specifies the location of the pedestrian at the time of the accident.
- Pedestrian_Movement: Describes the movement of the pedestrian during the accident.
- Car_Passenger, Bus_or_Coach_Passenger: Indicates if the casualty was a passenger in a car, bus, or coach.
- Pedestrian_Road_Maintenance_Worker: Specifies if the casualty was a road maintenance worker.
- Casualty_Type: Describes the type of casualty (e.g., driver, passenger, pedestrian).
- Casualty_Home_Area_Type: Specifies the type of area (urban, rural) where the casualty resides.

Vehicles Dataset

Vehicles entity set

The Vehicles entity set includes the following attributes:

- Accident_Index: A foreign key linking to the Accidents entity set.
- Vehicle_Reference: A unique identifier for each vehicle involved in an accident (Primary Key).
- Vehicle_Type: Specifies the type of the vehicle (e.g., car, motorcycle).
- Vehicle_Location_Restricted_Lane: Describes the location of the vehicle in a restricted lane.
- Junction_Location: Indicates the location of the vehicle concerning a junction.
- Was_Vehicle_Left_Hand_Drive: Specifies whether the vehicle was left-hand drive.
- Engine_Capacity_CC: Represents the engine capacity of the vehicle in cubic centimeters.
- Propulsion_Code: Specifies the propulsion code of the vehicle.
- Age_of_Vehicle: Represents the age of the vehicle.

Vehicles Sub-classifications

A subset of attributes from the Vehicles dataset was removed and split into 2 new entity sets:

Driver entity set: Attributes related to the driver of the vehicle.

Events entity set: Attributes related to events involving the vehicle.

Weak Entity Sets

1. Outcome - Accidents has a 1:1 participation with Outcome. Casualties has a 1:N participation with Outcome. This is because every accident has exactly one (1:1) outcome but every casualty can be a result of any number of accidents (1:N). Because of the 1:1 relation with Accidents, it does not require a separate schema.

2. Impacted - Accidents has a 1:1 participation with Impacted. Vehicles has a 1:N participation with Impacted. This is because every accident has exactly one (1:1) impact. However, there may be 1 or N vehicles in such impacts. Because of the 1:1 relation with Accidents, it does not require a separate schema.

3. Operator - Vehicles has a 1:1 participation with Operator. Driver has a 1:1 participation with Operator. This is because every vehicle has exactly one (1:1) driver and every driver can only operate exactly one (1:1) vehicle at the time of an accident. Because of the 1:1 relation with Vehicles and Driver, it does not require a separate schema.

4. Events - Vehicles has a 1:1 participation with Happened. Events has a 1:1 participation with Happened. This is because every vehicle has exactly one (1:1) specific event or accident facts. Similarly, every event can only correspond to one (1:1) vehicle at the time of an accident. Because of the 1:1 relation with Vehicles and Events, it does not require a separate schema.

Conclusion

The ER model design exhibits a normalized structure, facilitating efficient querying and maintenance of the database. Logical groupings of entities and attributes, along with the use of foreign keys, contribute to data integrity and consistency.

Table of Entity Sets, Attributes, and Primary Keys

Entity Set	Attributes	Primary Key
Accidents	Accident_Index Accident_Severity Number_of_Vehicles Number_of_Casualties	Accident Index
Casualties	Accident_Index Vehicle_Reference Casualty_Reference Casualty_Class Sex_of_Casualty Age_of_Casualty Age_Band_of_Casualty Casualty_Severity Pedestrian_Location Pedestrian_Movement Car_Passenger Bus_or_Coach_Passenger Pedestrian_Road_Maintenance_Worker Casualty_Type Casualty_Home_Area_Type	Accident_Index Casualty_Reference
Vehicles	Accident_Index Vehicle_Reference	Accident_Index Vehicle_Reference

	Vehicle_Type Vehicle_Location_Restricted_Lane Junction_Location Was_Vehicle_Left_Hand_Drive Engine_Capacity_CC Propulsion_Code Age_of_Vehicle	
Road	Accident_Index 1st_Road_Class 1st_Road_Number Road_Type Speed_limit 2nd_Road_Class 2nd_Road_Number	
Location	Accident_Index Location_Easting_OSGR Location_Northing_OSGR Longitude Latitude Junction_Detail Junction_Control Pedestrian_Crossing_Human_Control Pedestrian_Crossing_Physical_Facilities Urban_or_Rural_Area LSOA_of_Accident_Location	
Administration	Accident_Index Police_Force int Local_Authority_District Local_Authority_Highway Did_Police_Officer_Attend_Scene_of_Accident	
Timestamp	Accident_Index Date date Day_of_Week Time	
Conditions	Accident_Index Light_Conditions Weather_Conditions Road_Surface_Conditions Special_Conditions_at_Site Carriageway_Hazards	
Driver	Accident_Index Journey_Purpose_of_Driver Sex_of_Driver Age_of_Driver Age_Band_of_Driver Driver_IMD_Decile Driver_Home_Area_Type	
Events	Accident_Index Towing_and_Articulation Vehicle_Manoeuvre Skidding_and_Overturning Hit_Object_in_Carriageway Vehicle_Leaving_Carriageway Hit_Object_off_Carriageway 1st_Point_of_Impact	

Relationship Sets

1. Circumstances - Accidents has a 1:1 participation with Circumstances. Conditions has a 1:1 participation with Circumstances. This is because every accident has exactly one (1:1) circumstance associated with the environmental conditions. Similarly, every condition can be related to only one circumstance(1:1).

2. Where - Accidents has a 1:1 participation with Where. Location has a 1:1 participation with Where. This is because every accident has exactly one (1:1) location associated with it. Similarly, every location has a direct (1:1) relation with an accident.

3. When - Accidents has a 1:1 participation with When. Timestamp has a 1:1 participation with When. This is because every accident has exactly one (1:1) timestamp associated with it. Similarly, every timestamp has a direct (1:1) relation with an accident.

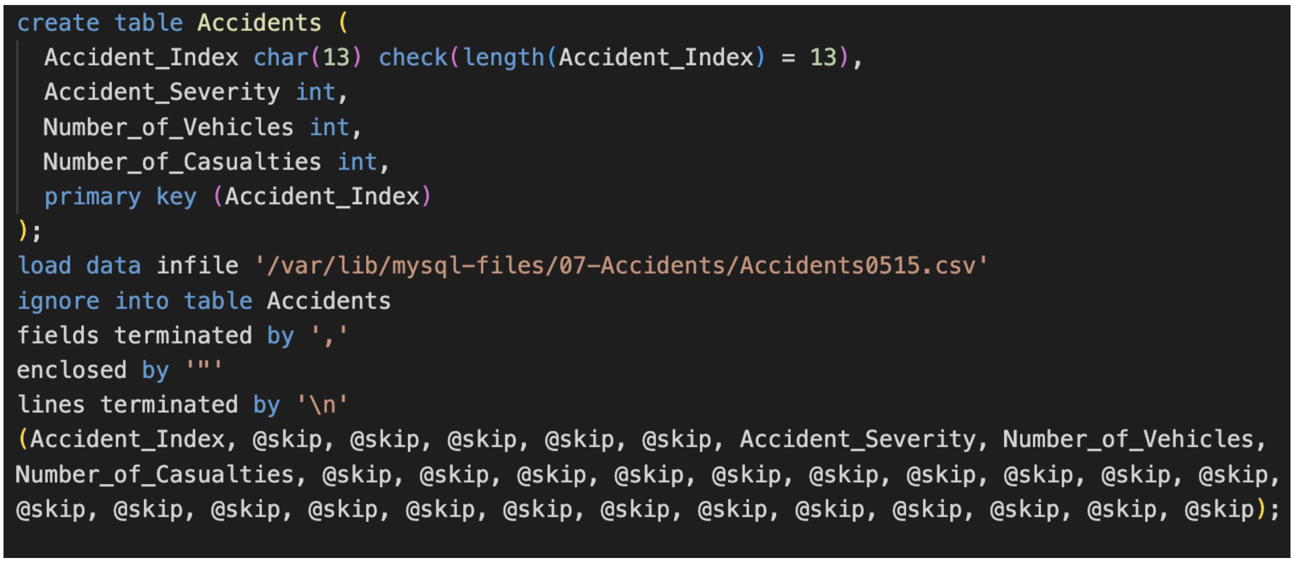
4. Enforcement - Accidents has a 1:1 participation with Enforcement. Administration has a 1:1 participation with Enforcement. This is because every accident has exactly one (1:1) Administrative Enforcement associated with it. Similarly, every enforcement set has a direct (1:1) relation with an accident.

5. Type - Location has a 1:1 participation with Type. Road has a 1:1 participation with Type. This is because every location is associated with exactly one (1:1) set of road characteristics. Similarly, every road set is associated with only one (1:1) location set.

Relational Schema

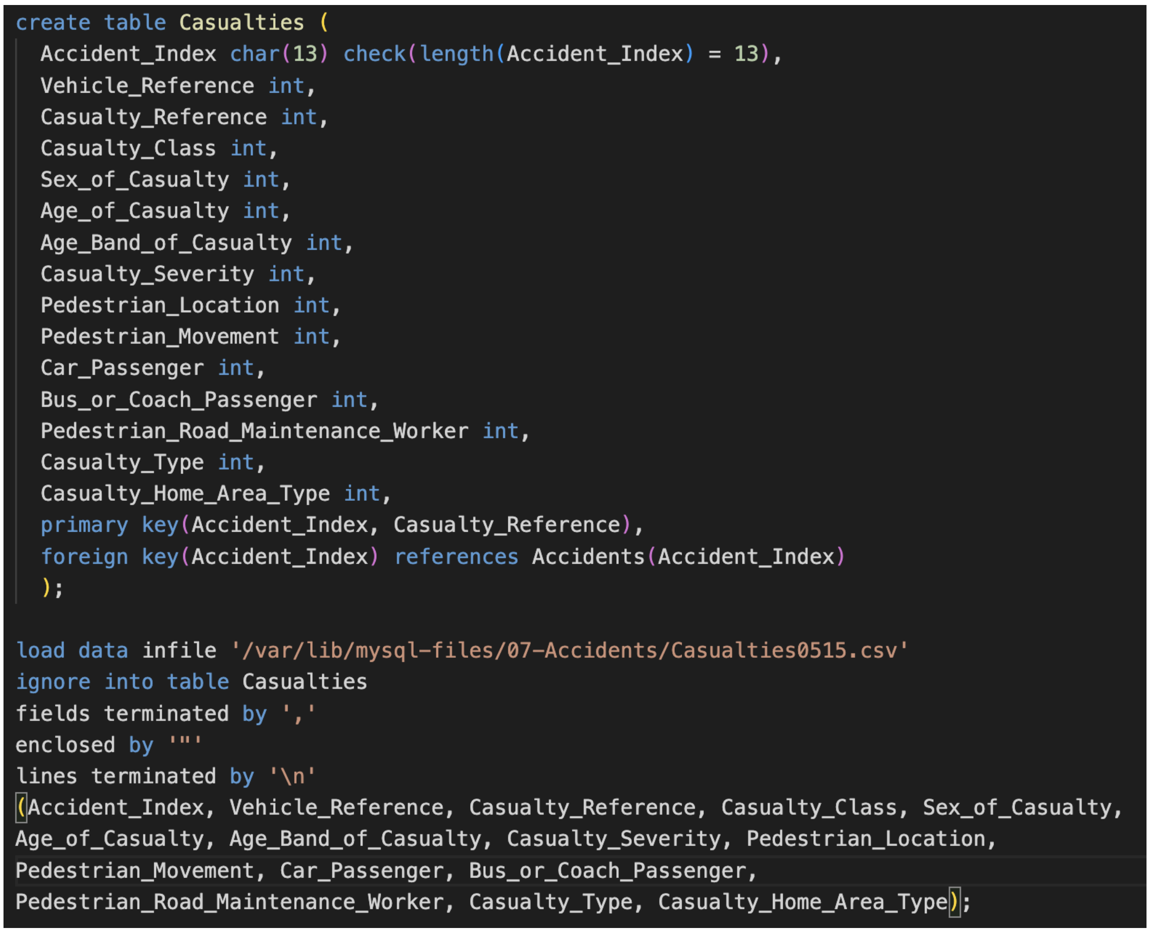
Table Creation and Data Population Code

1. Accidents



```
create table Accidents (
  Accident_Index char(13) check(length(Accident_Index) = 13),
  Accident_Severity int,
  Number_of_Vehicles int,
  Number_of_Casualties int,
  primary key (Accident_Index)
);
load data infile '/var/lib/mysql-files/07-Accidents/Accidents0515.csv'
ignore into table Accidents
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
(Accident_Index, @skip, @skip, @skip, @skip, @skip, Accident_Severity, Number_of_Vehicles,
Number_of_Casualties, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip,
@skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip);
```

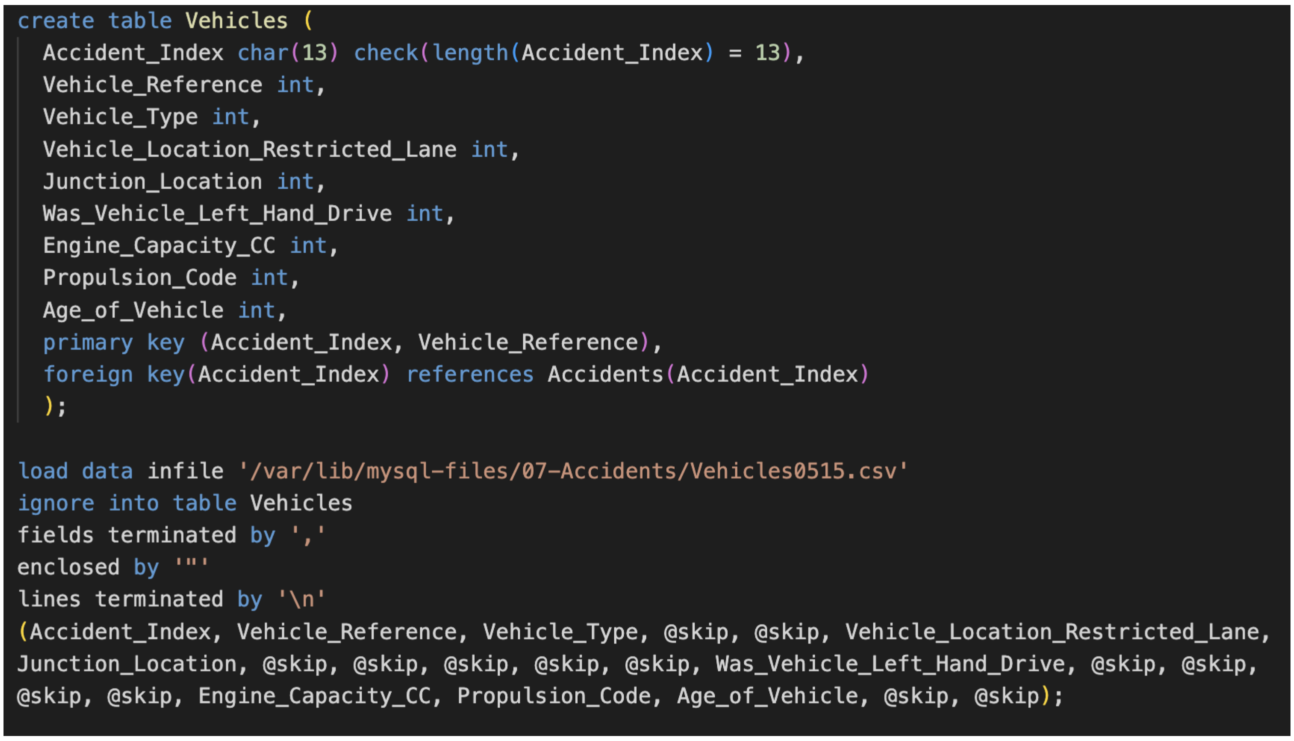
2. Casualties



```
create table Casualties (
  Accident_Index char(13) check(length(Accident_Index) = 13),
  Vehicle_Reference int,
  Casualty_Reference int,
  Casualty_Class int,
  Sex_of_Casualty int,
  Age_of_Casualty int,
  Age_Band_of_Casualty int,
  Casualty_Severity int,
  Pedestrian_Location int,
  Pedestrian_Movement int,
  Car_Passenger int,
  Bus_or_Coach_Passenger int,
  Pedestrian_Road_Maintenance_Worker int,
  Casualty_Type int,
  Casualty_Home_Area_Type int,
  primary key(Accident_Index, Casualty_Reference),
  foreign key(Accident_Index) references Accidents(Accident_Index)
  );

load data infile '/var/lib/mysql-files/07-Accidents/Casualties0515.csv'
ignore into table Casualties
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
(Accident_Index, Vehicle_Reference, Casualty_Reference, Casualty_Class, Sex_of_Casualty,
Age_of_Casualty, Age_Band_of_Casualty, Casualty_Severity, Pedestrian_Location,
Pedestrian_Movement, Car_Passenger, Bus_or_Coach_Passenger,
Pedestrian_Road_Maintenance_Worker, Casualty_Type, Casualty_Home_Area_Type);
```

We chose (Accident_Index, Casualty_Reference) as the primary key as it seemed the most appropriate choice for the Casualties dataset.

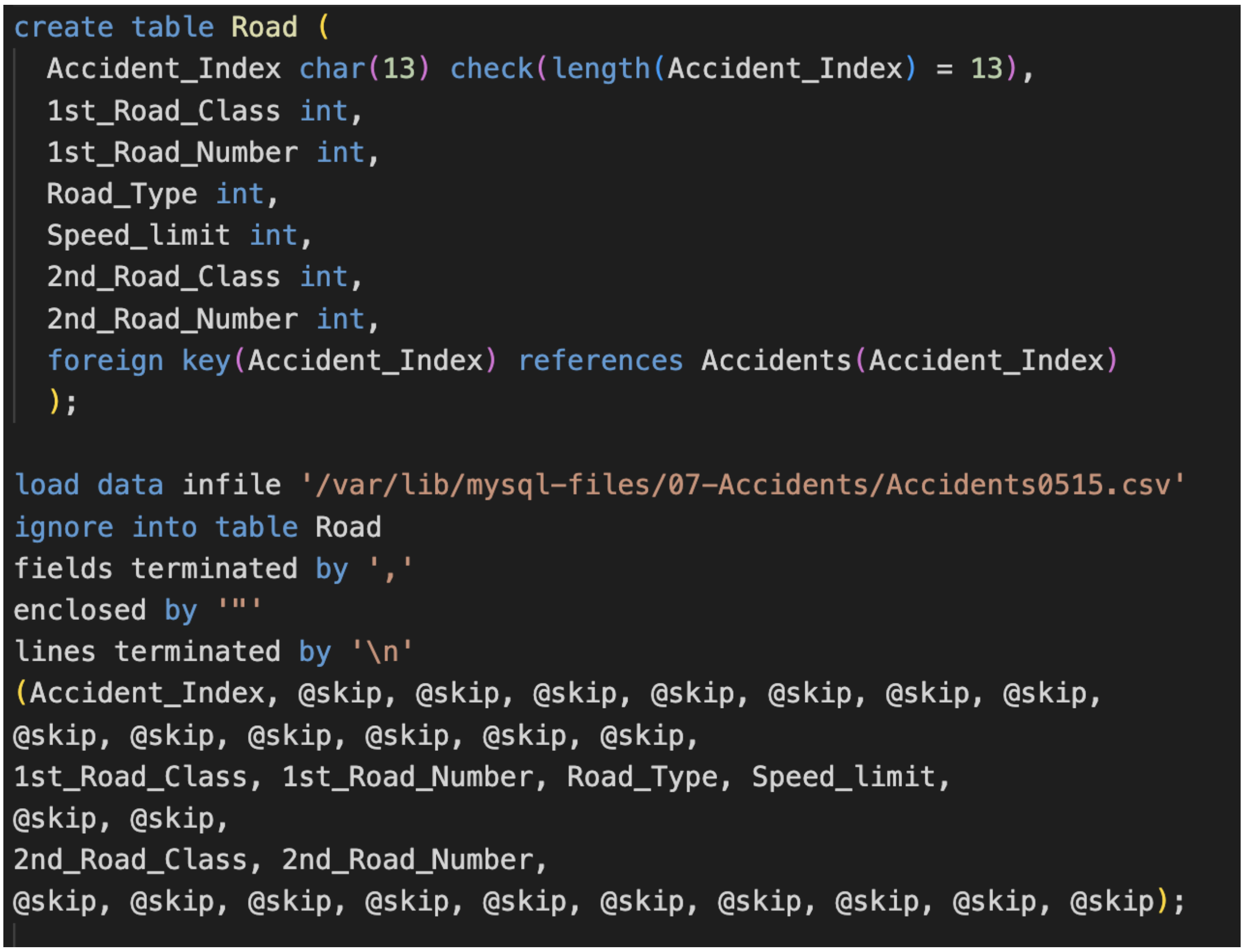
3. Vehicles



```sql
create table Vehicles (
  Accident_Index char(13) check(length(Accident_Index) = 13),
  Vehicle_Reference int,
  Vehicle_Type int,
  Vehicle_Location_Restricted_Lane int,
  Junction_Location int,
  Was_Vehicle_Left_Hand_Drive int,
  Engine_Capacity_CC int,
  Propulsion_Code int,
  Age_of_Vehicle int,
  primary key (Accident_Index, Vehicle_Reference),
  foreign key(Accident_Index) references Accidents(Accident_Index)
  );

load data infile '/var/lib/mysql-files/07-Accidents/Vehicles0515.csv'
ignore into table Vehicles
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
(Accident_Index, Vehicle_Reference, Vehicle_Type, @skip, @skip, Vehicle_Location_Restricted_Lane,
Junction_Location, @skip, @skip, @skip, @skip, @skip, Was_Vehicle_Left_Hand_Drive, @skip, @skip,
@skip, @skip, Engine_Capacity_CC, Propulsion_Code, Age_of_Vehicle, @skip, @skip);
```

We chose (Accident_Index, Vehicle_Reference) as the primary key as it seemed the most appropriate choice for the Vehicles dataset.

4. Road



```sql
create table Road (
  Accident_Index char(13) check(length(Accident_Index) = 13),
  1st_Road_Class int,
  1st_Road_Number int,
  Road_Type int,
  Speed_limit int,
  2nd_Road_Class int,
  2nd_Road_Number int,
  foreign key(Accident_Index) references Accidents(Accident_Index)
  );

load data infile '/var/lib/mysql-files/07-Accidents/Accidents0515.csv'
ignore into table Road
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
(Accident_Index, @skip, @skip, @skip, @skip, @skip, @skip, @skip,
@skip, @skip, @skip, @skip, @skip, @skip,
1st_Road_Class, 1st_Road_Number, Road_Type, Speed_limit,
@skip, @skip,
2nd_Road_Class, 2nd_Road_Number,
@skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip);
```

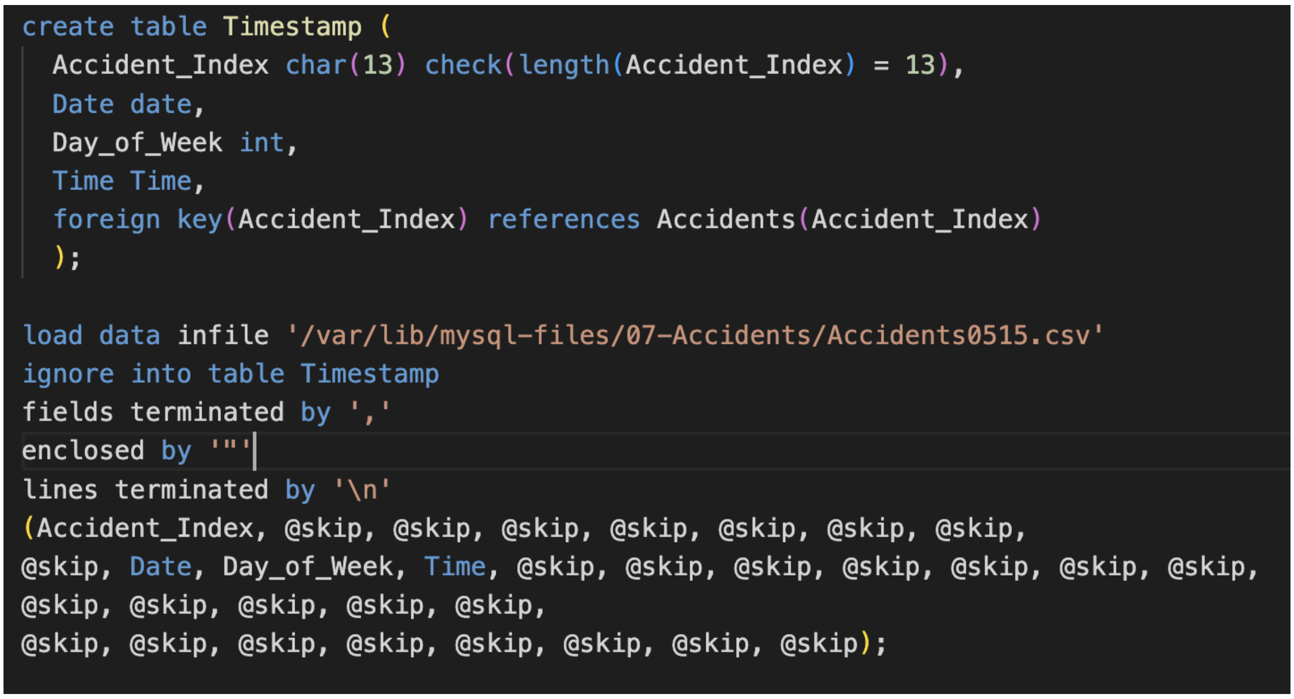
5. Location

```sql
create table Location (
  Accident_Index char(13) check(length(Accident_Index) = 13),
  Location_Easting_OSGR int,
  Location_Northing_OSGR int,
  Longitude float(6),
  Latitude float(6),
  Junction_Detail int,
  Junction_Control int,
  Pedestrian_Crossing_Human_Control int,
  Pedestrian_Crossing_Physical_Facilities int,
  Urban_or_Rural_Area int,
  LSOA_of_Accident_Location char(9),
  foreign key(Accident_Index) references Accidents(Accident_Index)
  );

load data infile '/var/lib/mysql-files/07-Accidents/Accidents0515.csv'
ignore into table Location
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
(Accident_Index, Location_Easting_OSGR, Location_Northing_OSGR, Longitude, Latitude, @skip, @skip,
@skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, Junction_Detail,
Junction_Control, @skip, @skip, Pedestrian_Crossing_Human_Control,
Pedestrian_Crossing_Physical_Facilities,
@skip, @skip, @skip, @skip, @skip, Urban_or_Rural_Area, @skip, LSOA_of_Accident_Location);
```

6. Administration

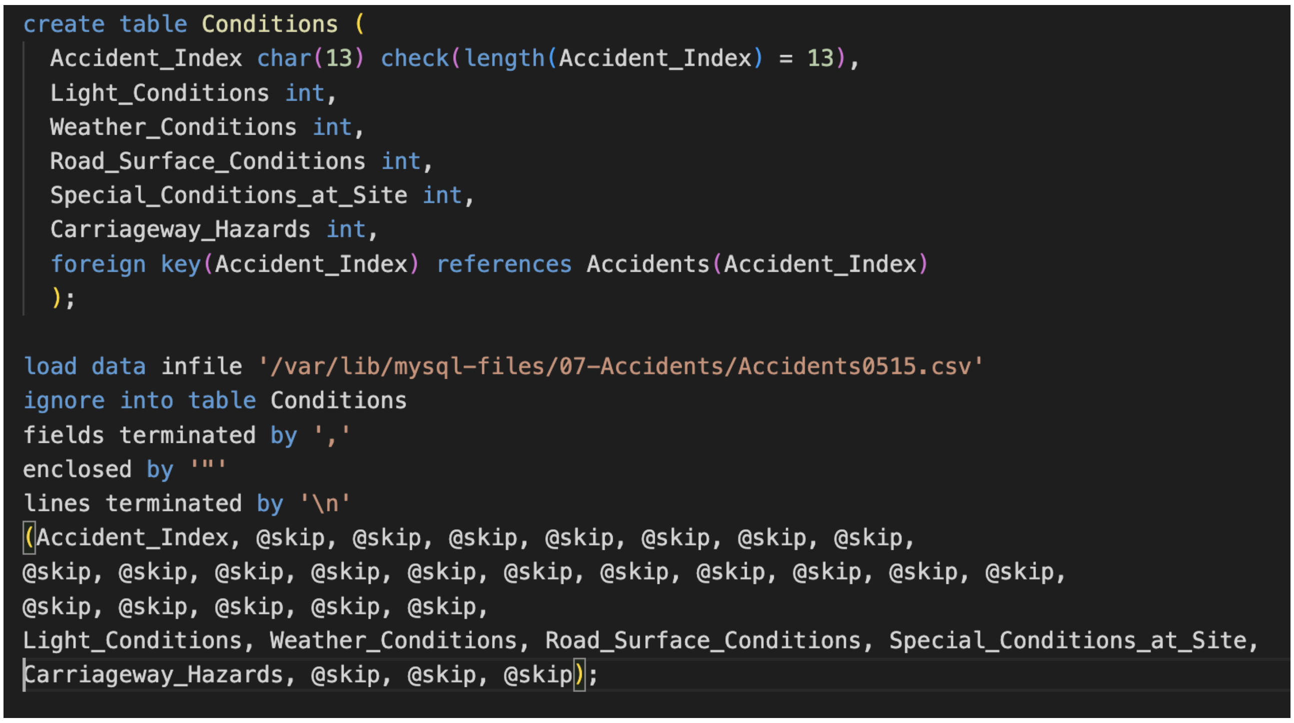
```sql
create table Administration (
   Accident_Index char(13) check(length(Accident_Index) = 13),
   Police_Force int,
   Local_Authority_District int,
   Local_Authority_Highway char(9),
   Did_Police_Officer_Attend_Scene_of_Accident int,
   foreign key(Accident_Index) references Accidents(Accident_Index)
  );

load data infile '/var/lib/mysql-files/07-Accidents/Accidents0515.csv'
ignore into table Administration
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
(Accident_Index, @skip, @skip, @skip, @skip, Police_Force, @skip, @skip,
@skip, @skip, @skip, @skip, Local_Authority_District, Local_Authority_Highway, @skip, @skip,
@skip, @skip, @skip,
@skip, @skip, @skip, @skip, @skip,
@skip, @skip, @skip, @skip, @skip, @skip, Did_Police_Officer_Attend_Scene_of_Accident, @skip);
```

7. Timestamp



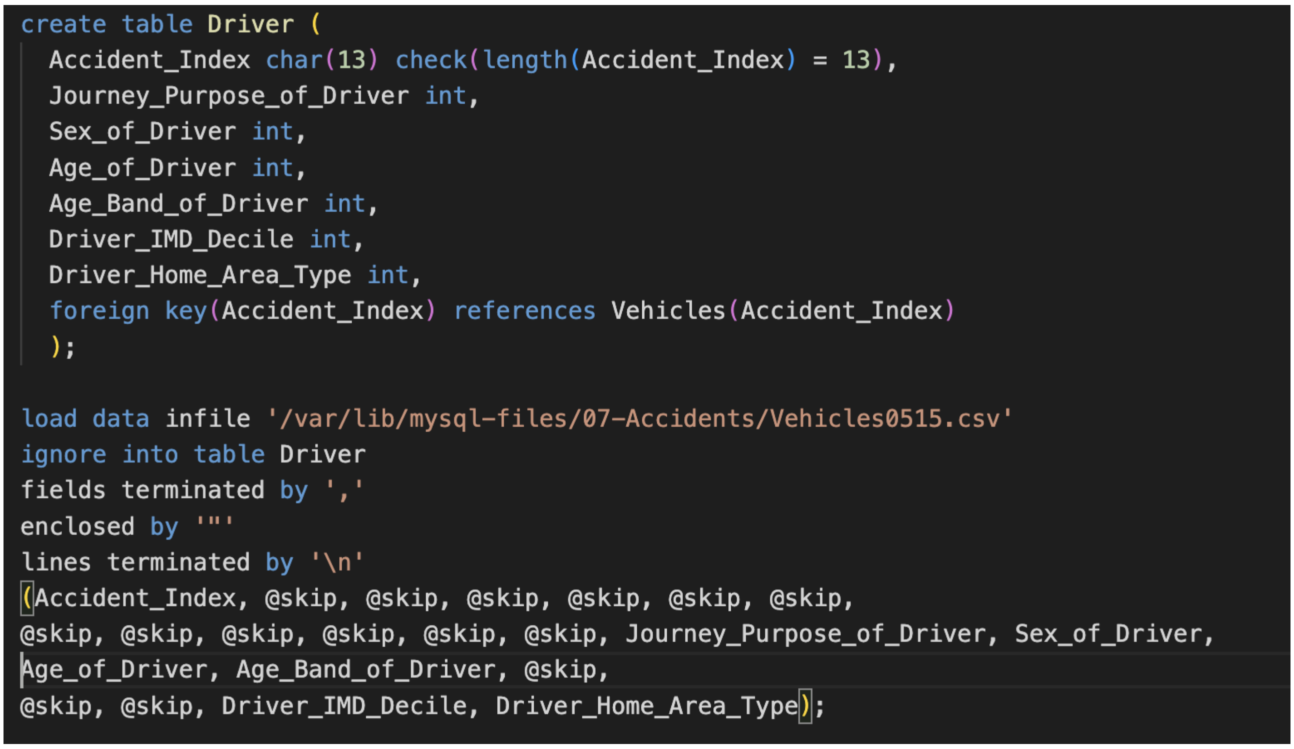
```sql
create table Timestamp (
  Accident_Index char(13) check(length(Accident_Index) = 13),
  Date date,
  Day_of_Week int,
  Time Time,
  foreign key(Accident_Index) references Accidents(Accident_Index)
  );

load data infile '/var/lib/mysql-files/07-Accidents/Accidents0515.csv'
ignore into table Timestamp
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
(Accident_Index, @skip, @skip, @skip, @skip, @skip, @skip, @skip,
@skip, Date, Day_of_Week, Time, @skip, @skip, @skip, @skip, @skip, @skip, @skip,
@skip, @skip, @skip, @skip, @skip,
@skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip);
```

8. Conditions



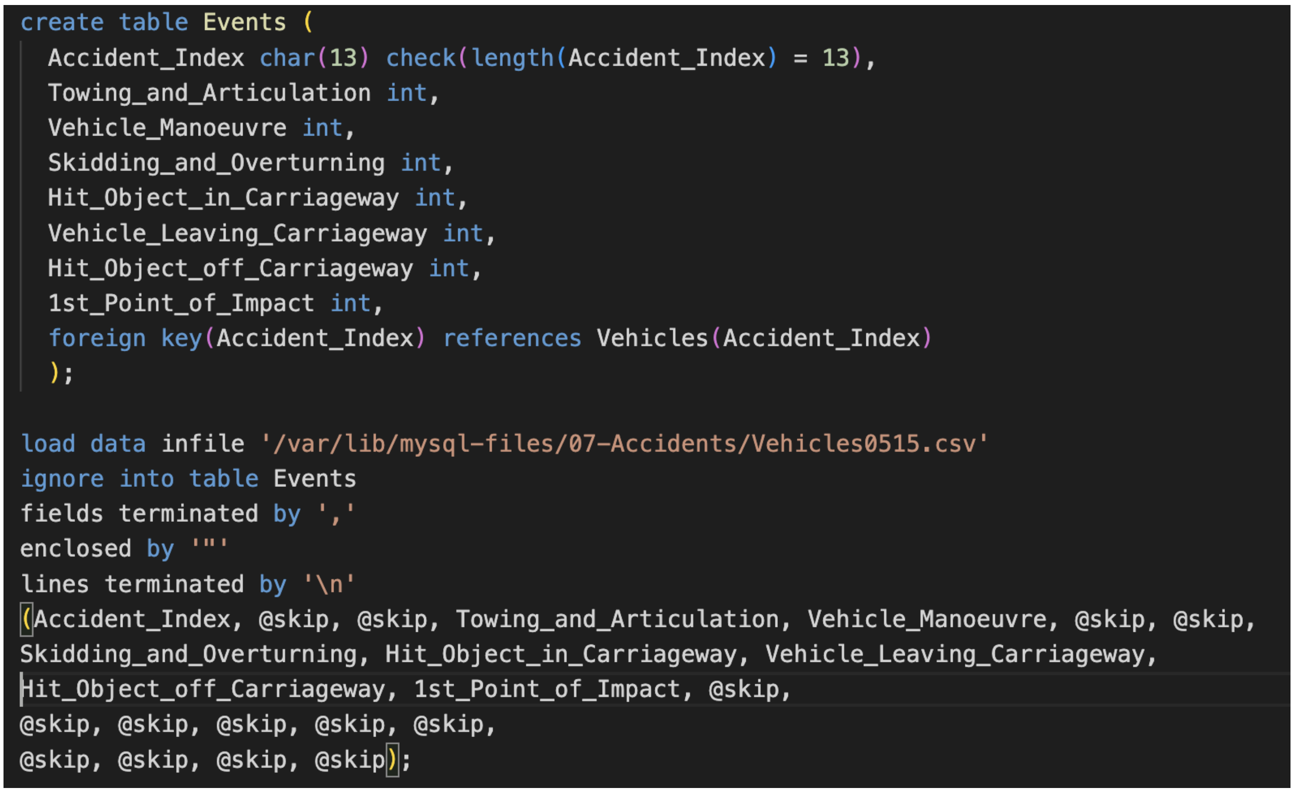
```sql
create table Conditions (
  Accident_Index char(13) check(length(Accident_Index) = 13),
  Light_Conditions int,
  Weather_Conditions int,
  Road_Surface_Conditions int,
  Special_Conditions_at_Site int,
  Carriageway_Hazards int,
  foreign key(Accident_Index) references Accidents(Accident_Index)
  );

load data infile '/var/lib/mysql-files/07-Accidents/Accidents0515.csv'
ignore into table Conditions
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
(Accident_Index, @skip, @skip, @skip, @skip, @skip, @skip, @skip,
@skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip, @skip,
@skip, @skip, @skip, @skip, @skip,
Light_Conditions, Weather_Conditions, Road_Surface_Conditions, Special_Conditions_at_Site,
Carriageway_Hazards, @skip, @skip, @skip);
```

9. Driver



```
create table Driver (
    Accident_Index char(13) check(length(Accident_Index) = 13),
    Journey_Purpose_of_Driver int,
    Sex_of_Driver int,
    Age_of_Driver int,
    Age_Band_of_Driver int,
    Driver_IMD_Decile int,
    Driver_Home_Area_Type int,
    foreign key(Accident_Index) references Vehicles(Accident_Index)
    );

load data infile '/var/lib/mysql-files/07-Accidents/Vehicles0515.csv'
ignore into table Driver
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
(Accident_Index, @skip, @skip, @skip, @skip, @skip, @skip,
@skip, @skip, @skip, @skip, @skip, @skip, Journey_Purpose_of_Driver, Sex_of_Driver,
Age_of_Driver, Age_Band_of_Driver, @skip,
@skip, @skip, Driver_IMD_Decile, Driver_Home_Area_Type);
```

10. Events



```
create table Events (
    Accident_Index char(13) check(length(Accident_Index) = 13),
    Towing_and_Articulation int,
    Vehicle_Manoeuvre int,
    Skidding_and_Overturning int,
    Hit_Object_in_Carriageway int,
    Vehicle_Leaving_Carriageway int,
    Hit_Object_off_Carriageway int,
    1st_Point_of_Impact int,
    foreign key(Accident_Index) references Vehicles(Accident_Index)
    );

load data infile '/var/lib/mysql-files/07-Accidents/Vehicles0515.csv'
ignore into table Events
fields terminated by ','
enclosed by '"'
lines terminated by '\n'
(Accident_Index, @skip, @skip, Towing_and_Articulation, Vehicle_Manoeuvre, @skip, @skip,
Skidding_and_Overturning, Hit_Object_in_Carriageway, Vehicle_Leaving_Carriageway,
Hit_Object_off_Carriageway, 1st_Point_of_Impact, @skip,
@skip, @skip, @skip, @skip, @skip,
@skip, @skip, @skip, @skip);
```

While loading the data from all the CSVs, we use the ‘ignore’ keyword to skip the insertion of rows with errors. This helps prevent many types of errors arising from incorrect data formats like incorrect time format, rows with more data than columns, and type mismatches. We also implement a check on Accident_Index to prevent rows with incorrect indices and ‘Accident_Index’ values for Accident_Index.

During the testing phase, we deliberately inserted data into the database using different scenarios to evaluate the handling of various situations by the code. This included inserting rows with the exact number of values expected by the columns, as well as cases where there were more or fewer values. By doing so, we aimed to verify the code's ability to handle different input scenarios gracefully.

One crucial aspect of our testing involved intentionally introducing incorrect data types in the insert statements. This helped us assess the code's ability to detect and handle data type mismatches, ensuring that the database remains consistent and accurate. Detecting and handling such errors robustly is essential for maintaining data integrity and preventing issues that could arise from incompatible data types.

The mixed approach of inserting both correct and incorrect data allowed us to observe how the code responded to various situations. By intentionally introducing errors, we could simulate real-world scenarios where users might make mistakes or provide inconsistent data.

The result of this comprehensive testing was that the code showed efficiency in handling different input scenarios. The code was able to detect and appropriately respond to issues such as incorrect data types, ensuring that the database remained consistent and error-free.

Data Mining Investigations

1. Domain-appropriate question

What factors (specifically that of the environment or person) can best predict the severity of an accident (**before** the accident occurs)?

2. Techniques to solve the problem

Since this is a classification problem, we will use classification techniques to solve the problem. Specifically, we'll use a validated decision tree. Therefore, we need to pick out relevant attributes. I carefully chose a list of attributes that I thought could be related to the severity of an accident:

~~Date (binned into seasons)~~, Time, Road_Type, ~~Junction_Detail~~, Junction_Control, Speed_limit, Light_Conditions, Weather_Conditions, Road_Surface_Conditions, ~~Urban_or_Rural_Area~~, Vehicle_Type, Vehicle_Manoeuvre, ~~Junction_Location~~, Journey_Purpose_of_Driver, Age_Band_of_Driver, Age_of_Vehicle

Through trial and error, I reduced the number of attributes to those that produced a higher accuracy. For Speed_limit and attributes that are not already binned, I binned them and denoted them with a * in front of the attribute below.

<u>Accidents Table:</u>

- **Accident_Severity**: This is the attribute I'm trying to predict (1: Fatal, 2: Serious, 3: Slight)
- ***Time**: The of the day - morning, noon, etc.
- **Road_Type**: The type of road - roundabout, oneway, etc.
- **Junction_Control**: Details of the junction - stop light, stop sign, etc.
- ***Speed_limit**: The speed limit - higher speeds could lead to more severe accidents
- **Light_Conditions**, **Weather_Conditions**, **Road_Surface_Conditions**: The environmental factors

Note: we won't use attributes like **Number_of_Vehicles (involved)** as we're specifically looking at attributes **before** the accident occurs.

<u>Vehicles Table:</u>

- **Vehicle_Type**: The type of vehicle - car, van, motorcycle, etc.
- **Vehicle_Manoeuvre**: The maneuver the vehicle took
- **Journey_Purpose_of_Driver**: The purpose of driving
- **Age_Band_of_Driver**: The age of the driver (instead of Age_of_Driver to eliminate the binning step)
- ***Age_of_Vehicle**: Age of the vehicle

Note: we won't use attributes like **Skidding_and_Overturning**, **Hit_Object_in_Carriageway**, **Hit_Object_off_Carriageway** as we're specifically looking at attributes **before** the accident occurs.

<u>Casualty Table:</u>

- **No Attributes**

Note: we won't use attributes from the casualty table as we don't want to use data from hindsight.

Furthermore, I did not remove entries with "unknown" values as it could lead to statistical bias. For instance, accidents with unknown details about the driver could relate more to severe accidents.

3. Implementing the technique (continued in 4.)

All the code for implementation is in the GitLab repository. For preprocessing, most attributes were already binned and the distribution of the bins for each attribute was fair (except for Speed_limit). I created a Python script `model.py` to retrieve, aggregate, and preprocess the attributes above from the SQL database. For preprocessing, the script categorized **Time** into 5 periods:

1. Early Morning - 0 to 5 hours
2. Morning: 5 to 12 hours
3. Afternoon: 12 to 17 hours
4. Evening: 17 to 21 hours
5. Night: 21 to 24 hours

and **Age_of_Vehicle** into 5 bins:

-1. Unknown
1. New: Less than or equal to 3 years
2. Fairly New: More than 3 years and up to 7 years
3. Old: More than 7 years and up to 15 years
4. Very Old: More than 15 years

After some tuning, I found that selecting these 5 particular bins and categorizing them in this way (morning, afternoon, etc. + new, fairly new, old, etc.) helped the model be more accurate.

For Speed_limit, I binned 15, 10, and 0 with 20 since their sample size was extremely small to prevent them from having an outsized influence on the model or causing overfitting. If the sample size was bigger, I would have kept them in their own category.

Before training the data, I noticed that the distribution of the target variable (Accident_Severity) had a very imbalanced frequency distribution:

1.	Fatal	(~1%)
2.	Serious	(~14%)
3.	Slight	(~85%)

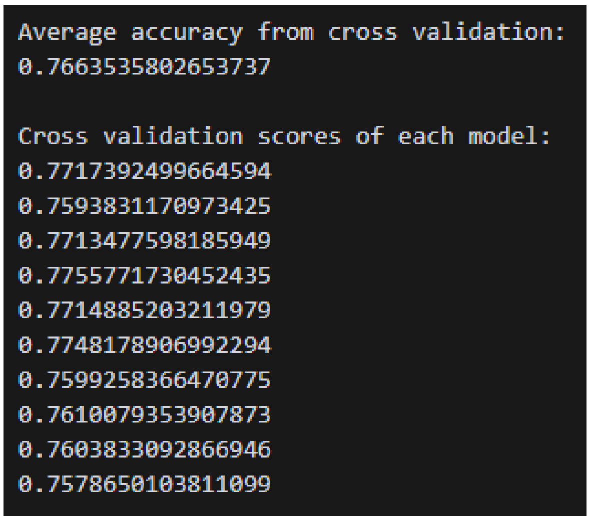
This can lead to a machine learning model being biased towards the majority class (3. Slight), resulting in poor performance when predicting the minority class instances. Because of this, I added an oversampler from the imbalanced-learn library to rectify the percent frequency to be around 33.33% for each severity. Although this led to an overall lower accuracy rating, this improves the model's precision in predicting the minority severities (fatal and serious.)

After I resampled the data, I trained and tested the model with `DecisionTreeClassifier`. Since the DecisionTreeClassifier is ordinal by default, I binned each attribute with `LabelEncoder`. Instead of training and testing the data with `fit()` and `predict()`, I did the training and testing using `cross_val_score()`, which also trains and tests the model.

4. Determining the validity of the model (continued from 3.)

I used k-fold cross-validation, specifically with the `cross_val_score()` function to determine the validity of my model. I used a fold value of 10 instead of 5 because the dataset is large. This gave me a better-performing model.

After I trained and tested the model, I printed the cross-validation scores of each set in `output.txt`. The output is shown below:

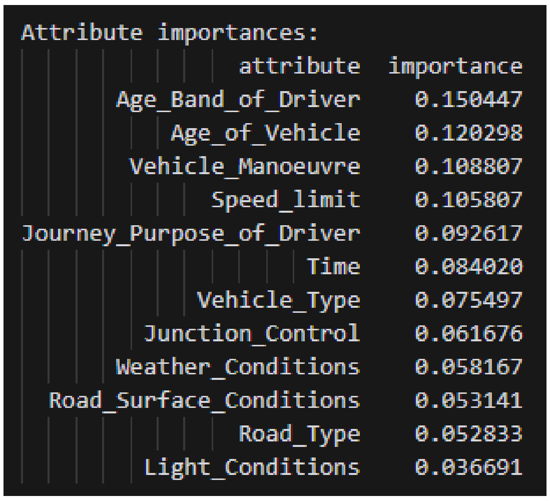


```
Average accuracy from cross validation:
0.7663535802653737

Cross validation scores of each model:
0.7717392499664594
0.7593831170973425
0.7713477598185949
0.7755771730452435
0.7714885203211979
0.7748178906992294
0.7599258366470775
0.7610079353907873
0.7603833092866946
0.7578650103811099
```

As you can see the average accuracy of the model is 0.766. Since the sample after oversampling had a distribution of ⅓ for each severity group, a random guess would result in an accuracy of ⅓. This means our model is better at predicting the accident severity before the accident occurs.

5. Results

For finding the most significant predictor of Accident_Severity, we would need to look at the splits of the trees using the Gini impurity and entropy values. Since the scikit-learn library already does that for me, I used the `feature_importances_` attribute and extracted the predictors of accident severity in descending order. The results are printed to `output.txt` and are shown below:



```
Attribute importances:
                 attribute  importance
          Age_Band_of_Driver    0.150447
              Age_of_Vehicle    0.120298
           Vehicle_Manoeuvre    0.108807
                 Speed_limit    0.105807
  Journey_Purpose_of_Driver    0.092617
                        Time    0.084020
                Vehicle_Type    0.075497
            Junction_Control    0.061676
          Weather_Conditions    0.058167
     Road_Surface_Conditions    0.053141
                   Road_Type    0.052833
            Light_Conditions    0.036691
```

Conclusion
To formally answer the data mining question I had at the beginning, the best predictors of accident severity in descending order of importance are: Age_Band_of_Driver, Age_of_Vehicle, Vehicle_Manoeuvre, Speed_limit, Journey_Purpose_of_Driver, Time, Vehicle_Type, Junction_Control, Weather_Conditions, Road_Surface_Conditions, Road_Type, Light_Conditions.

Other/Extra: values of attributes related to more severe accidents

I made a script, `average_severity.py`, that prints out the weighted average severities (fatal is given a weight of 10, severe is 3, slight is 1) of each value of the top 4 predictors. This helps us *generally* observe which values of the top 4 strongest predictors, found above, lead to more severe accidents. Note: this is not to say that all the values from the top 4 attributes are better predictors than all the values of other attributes; for instance, the age of the driver being 25 years old may tell us less about the possible accident severity than if the road is icy. The first column is the foreign key and the corresponding meaning can be found in `context.xlsx`.

Age_Band_of_Driver
Younger drivers could lack experience or thrillseek, which could lead to more severe accidents. On the other spectrum, older drivers could have physical impairments, which could also contribute to accident severity.

Using `average_severity.py`, I observed that the age groups most frequently involved in more severe accidents, listed in decreasing order of association with accident severity, are those aged 75 and above, followed by ages 66-75, and then ages 11-15. This makes sense for the reasons above while the healthy age groups from 26 to 45 are the least likely to be involved in more severe accidents.

Age_of_Vehicle
Older vehicles could lack the safety features that more modern cars have, which could lead to more severe accidents. They may also be more susceptible to mechanical failures, which could potentially lead to more severe accidents.

Using `average_severity.py`, I observed that the age band of vehicles most frequently involved in more severe accidents are cars in the Very Old (15+ years) category that I created which makes sense for the reasons above. The second was New (under 3 years) which makes sense as newer cars could be correlated to newer drivers, or if the driver is adjusting to their new car. The in-betweens were the least likely to be involved in more severe accidents.

Vehicle_Manoeuvre
Certain manoeuvres may be more dangerous to perform and therefore lead to more severe accidents.

Using `average_severity.py`, I observed that the top 2 most severe manoeuvres were overtaking which makes sense as it is a riskier manoeuvre. On the other side of the spectrum, the least severe ones were “Waiting to go - held up” and “Slowing or stopping” which also makes sense as slower speeds relate to less severe accidents.

Speed_Limit
Higher speed limits warrant faster driving speeds, which can lead to more severe crashes and therefore more severe accidents.

Using `average_severity.py`, I observed that the severity of speed limits in mph from most severe to least severe are 60, 50, 70, 40, 20, 30. This was generally expected but also surprising at first as I expected 70 to be the most severe and 20 to be further down. However, we need to consider that the relationship between speed limits and accident severity is also influenced by other factors. For instance, lower speed limits like 20 mph are often used in areas with high pedestrian activity. Furthermore, areas using 50 or 60 mph may have higher traffic density which could correlate to more severe accidents.