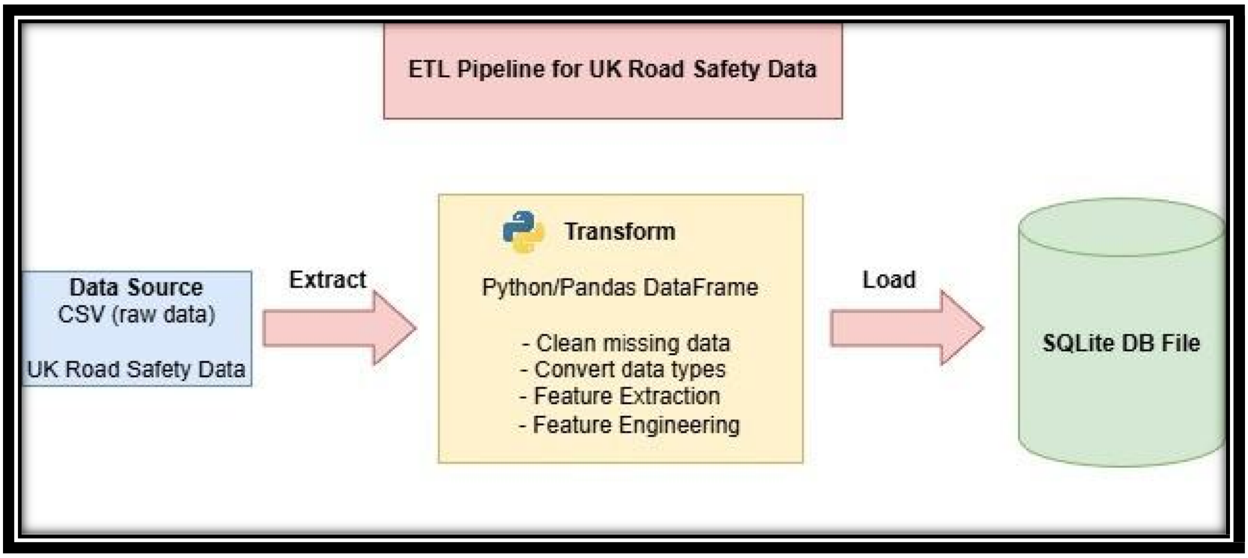
UK Road Safety Data Pipeline

BY DANIEL JONES

Table of Contents

Chapter 1: The Data Pipeline

Introduction



The aim of this data pipeline is to process raw data from the last 5 years and load it into a relational database for analysis. This data can then be used to assess risk exposure by certain attributes or develop predictive models for accident patterns in the UK. I will be working with UK Road Safety data from the Department of Transport (DfT). It is also assumed that the data going through this process will contain the same column names.

The code samples shown may not exactly match the final implementation in the script but perform the same operations. The shown images are provided for clarity and understanding and are representative of the actual code.

The python libraries we will be using are below. You may need to install these libraries before running the script.

Downloading libraries:

```
# Run if you need to install libraries required

"""
pip install pandas
pip install numpy
pip install matplotlib
pip install seaborn

"""
```

Importing libraries:

```
import pandas as pd                 # For Data Manipulation
import numpy as np                  # For Data Manipulation
import matplotlib.pyplot as plt     # For Data Visualisation
import seaborn as sns               # For Data Visualisation
import sqlite3                      # For SQL
import os                           # For Directory Management
```

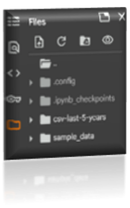
1.1 Data Extraction

The 3 CSV files I will be using are the casualty, collision and vehicle casualty statistics for the last 5 years (2020 – 2024) stored in a folder named "csv-last-5-years"

To make the main code cleaner, I created a function called "loadTransform" that takes the category name as a parameter, assuming the naming convention remains consistent. I chose to do this because the downloaded CSV files for the past 5 years follow the same pattern.

There is also an additional parameter that converts a columns data type to python string. This is because, when loading a CSV into a pandas DataFrame, some columns default to object dtype due to having multiple data types present. These tend to be identifiers so they will be changed to python string rather than pandas string type. However, this does mean they will still appear as dtype object but will be reflected in this documents table of Data types like this "object → string".

```python
# loadTransform function loads the data, and is also able to convert specified columns to python string

def loadTransform(category, object_columns=None):
  df = pd.read_csv(f"./csv-last-5-years/dft-road-casualty-statistics-{category}-last-5-years.csv")

  # Converts columns specified in parameters to python string (intended for columns of mixed types)

  if object_columns:
      for column in object_columns:
          df[column] = df[column].astype(str)

  return df
```



This code works correctly in Google Colab, as it sets the working directory automatically. In other environments such as Visual Studio, you may need to change your working directory to ensure the file paths work as intended. A potential fix not shown here has already been implemented into the function by importing "os" and adding a try and catch statement but I can't guarantee it will work in all cases.

Using the function (ignoring second parameter for now), the data is loaded into Pandas DataFrames for use in the next section, where the data exploration can begin.

```python
# Loading CSV

dfCasualty = loadTransform("casualty")
dfCollision = loadTransform("collision")
dfVehicle = loadTransform("vehicle")
```

1.2 Data Exploration

In this section I will be exploring each dataset in order to understand its structure and contents.

A lot of int data types represent categorical data and is outlined in Road safety open data guide provided by the DfT.

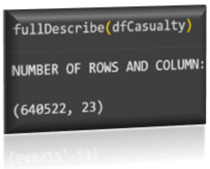
For each dataset I will be looking at:

(a) Dimension of data
(b) Attribute data types
(c) Number & percentage of missing values for each attribute
(d) Descriptive statistics for numeric and non-numeric attributes
(e) Two interesting visualisations

The following function ("fullDescribe") I created will be used to get the information of (a) to (d) for each dataset and will be later commented out in order to not affect the efficiency of the pipeline. Some environments require you to close the visualisation before the code continues.

```
# loadDescribe function: loads and describes the data

def fullDescribe(df):


  # (a) Dimension data

  print("NUMBER OF ROWS AND COLUMN: \n")
  print(df.shape) # Number of rows and columns
  print("\n")

  # (b) Attribute types

  print("COLUMNS AND DATA TYPES: \n")
  print(df.info()) # Columns and data types
  print("\n")

  # (c) Missing values

  # Making a dataframe of the columns and missing value/percentage

  print ("DATAFRAME OF MISSING: \n")
  df_replaceNA = df.replace([-1,"-1"],np.nan)
  dfMissing = pd.DataFrame({
    'Column Name': df_replaceNA.columns,
    'Missing Count': df_replaceNA.isnull().sum(),
    'Missing Percentage': (df_replaceNA.isnull().sum() / len(df_replaceNA) * 100).round(2)
    })
  print(dfMissing)
  print("\n")

  # (d) Descriptive statistics

  print("DESCRIPTIVE STATISTICS: \n")
  print(df_replaceNA.describe(include='all')) # Descriptive Statistics
  print("\n")

  return
```

Casualty Dataset



(a) In the Casualty dataset there is 640522 Rows (Observations) and 23 Columns (Attributes/Features)

(b) (c) Table of data types and Number & percentage of missing values

Column Index	Column Name	Data Type	Missing Values	% Missing Values
0	collision_index	object → string	0	0%
1	collision_year	int	0	0%
2	collision_ref_no	object → string	0	0%
3	vehicle_reference	int	0	0%
4	casualty_reference	int	0	0%
5	casualty_class	int	0	0%
6	sex_of_casualty	int	5740	≈ 0.90%
7	age_of_casualty	int	13846	≈ 2.16%
8	age_band_of_casualty	int	13846	≈ 2.16%
9	casualty_severity	int	0	≈ 0.002%
10	pedestrian_location	int	14	≈ 0.002%
11	pedestrian_movement	int	13	≈ 0.002%
12	car_passenger	int	2737	≈ 0.43%
13	bus_or_coach_passenger	int	255	≈ 0.04%
14	pedestrian_road_maintenance_worker	int	12077	≈ 1.89%
15	casualty_type	int	4935	≈ 0.77%
16	casualty_imd_decile	int	69483	≈ 10.85%
17	lsoa_of_casualty	object → string	91945	≈ 14.36%
18	enhanced_casualty_severity	int	271013	≈ 42.31%
19	casualty_injury_based	int	0	0%
20	casualty_adjusted_severity_serious	float	0	0%
21	casualty_adjusted_severity_slight	float	0	0%
22	casualty_distance_banding	int	529525	≈ 82.67%

(d) Below are the descriptive statistics for numeric and non-numeric attributes

	collision_index	collision_year	collision_ref_no	vehicle_reference \
count	640522	640522.000000	640522	640522.000000
unique	503475	NaN	501768	NaN
top	2023520300610	NaN	520300610	NaN
freq	70	NaN	70	NaN
mean	NaN	2022.047062	NaN	1.453419
std	NaN	1.388674	NaN	2.544908
min	NaN	2020.000000	NaN	1.000000
25%	NaN	2021.000000	NaN	1.000000
50%	NaN	2022.000000	NaN	1.000000
75%	NaN	2023.000000	NaN	2.000000
max	NaN	2024.000000	NaN	999.000000

	casualty_reference	casualty_class	sex_of_casualty	age_of_casualty
count	640522.000000	640522.000000	634782.000000	626676.000000
unique	NaN	NaN	NaN	NaN
top	NaN	NaN	NaN	NaN
freq	NaN	NaN	NaN	NaN
mean	1.347821	1.470640	1.382401	37.760409
std	2.612620	0.726339	0.493361	18.975233
min	1.000000	1.000000	1.000000	0.000000
25%	1.000000	1.000000	1.000000	23.000000
50%	1.000000	1.000000	1.000000	34.000000
75%	1.000000	2.000000	2.000000	51.000000
max	999.000000	3.000000	9.000000	101.000000

	age_band_of_casualty	casualty_severity	...	bus_or_coach_passenger \
count	626676.000000	640522.000000	...	640267.000000
unique	NaN	NaN	...	NaN
top	NaN	NaN	...	NaN
freq	NaN	NaN	...	NaN
mean	6.484028	2.785698	...	0.048714
std	2.229079	0.439563	...	0.426443
min	1.000000	1.000000	...	0.000000
25%	5.000000	3.000000	...	0.000000
50%	6.000000	3.000000	...	0.000000
75%	8.000000	3.000000	...	0.000000
max	11.000000	3.000000	...	9.000000

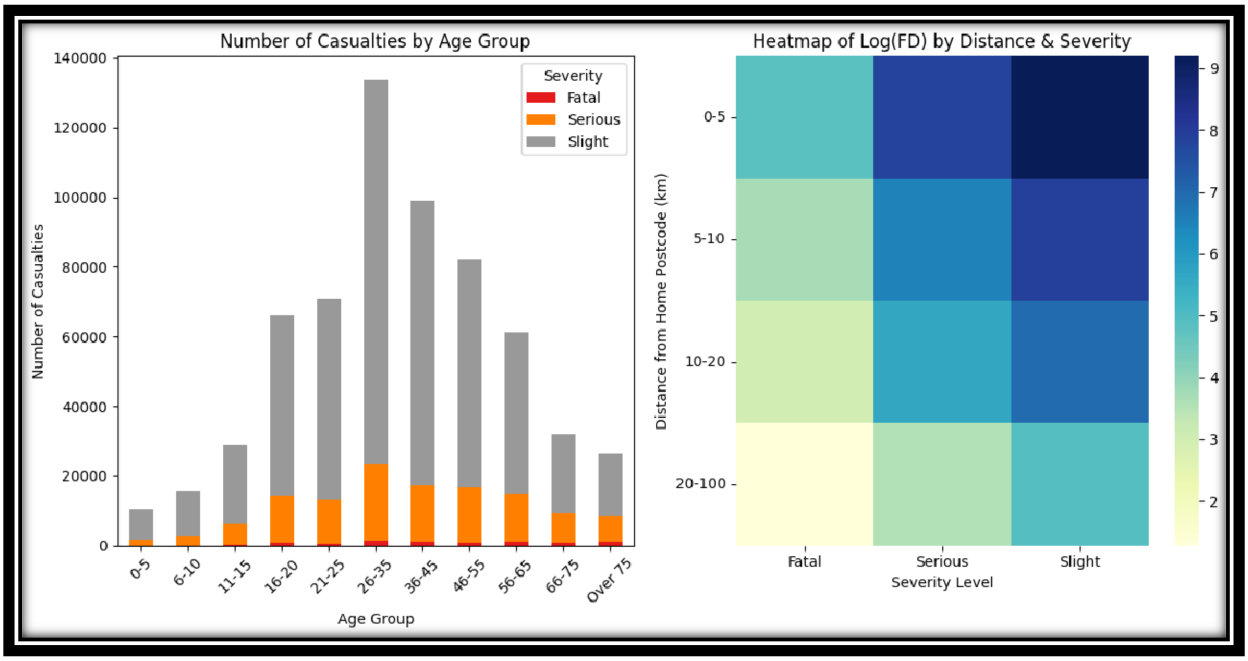
	pedestrian_road_maintenance_worker	casualty_type \
count	628445.000000	635587.000000
unique	NaN	NaN
top	NaN	NaN
freq	NaN	NaN
mean	0.024207	7.158247
std	0.215993	8.727578
min	0.000000	0.000000
25%	0.000000	1.000000
50%	0.000000	9.000000
75%	0.000000	9.000000
max	2.000000	99.000000

	casualty_imd_decile	lsoa_of_casualty	enhanced_casualty_severity \
count	571039.000000	548577	369509.000000
unique	NaN	36239	NaN
top	NaN	E01019456	NaN
freq	NaN	214	NaN
mean	4.939043	NaN	3.700281
std	2.793552	NaN	1.478695
min	1.000000	NaN	1.000000
25%	2.000000	NaN	3.000000
50%	5.000000	NaN	3.000000
75%	7.000000	NaN	3.000000
max	10.000000	NaN	7.000000

	casualty_injury_based	casualty_adjusted_severity_serious \
count	640522.000000	640522.000000
unique	NaN	NaN
top	NaN	NaN
freq	NaN	NaN
mean	0.576887	0.205310
std	0.494053	0.386169
min	0.000000	0.000000
25%	0.000000	0.000000
50%	1.000000	0.000000
75%	1.000000	0.098678
max	1.000000	1.000000

	casualty_adjusted_severity_slight	casualty_distance_banding
count	640522.000000	110997.000000
unique	NaN	NaN
top	NaN	NaN
freq	NaN	NaN
mean	0.782270	1.905772
std	0.395338	1.216647
min	0.000000	1.000000
25%	0.887130	1.000000
50%	1.000000	1.000000
75%	1.000000	3.000000
max	1.000000	5.000000

(e) Below are 2 interesting and meaningful visualisations for the Casualty dataset



This stacked column chart shows the number of casualties by age group and severity level.

Slight severity is the most common among all age groups however, ages 56+ have a higher proportion of fatal injuries which may be due to increase vulnerability.

Age group 26-35 have the most Casualties which could be due to more data recorded for them or suggest they are risker drivers. Younger children are likely with parents and older age groups likely drive less which may lower casualty rate overall for those groups.

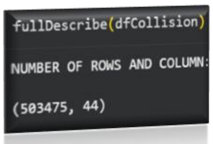
This graph may be used as insight for insurance quotes for different age groups.

This Heatmap shows the log transformed frequency density across severity levels and distance from home postcode. A darker colour indicates higher frequency density.

This is a quick visual way to see casualty differences by distance and severity level. This may be used for insurance company dashboards to see risk levels and justify price points based on distance travelled from home.

Frequency density was used due to unequal bin widths, and a log transformation was applied because certain categories dominate, making it harder to see the difference between groups

Collision Dataset


fullDescribe(dfCollision)
NUMBER OF ROWS AND COLUMN:
(503475, 44)

(a) In the Collision dataset there is 503475 Rows (Observations) and 44 Columns (Attributes/Features)

(b) (c) Table of data types and Number & percentage of missing values

Column Index	Column Name	Data Type	Missing Values	% Missing Values
0	collision_index	object → string	0	0%
1	collision_year	int	0	0%
2	collision_ref_no	object → string	0	0%
3	location_easting_osgr	float	65	≈ 0.01%
4	location_northing_osgr	float	65	≈ 0.01%
5	longitude	float	65	≈ 0.01%
6	latitude	float	65	≈ 0.01%
7	police_force	int	0	0%
8	collision_severity	int	0	0%
9	number_of_vehicles	int	0	0%
10	number_of_casualties	int	0	0%
11	date	object → datetime (pandas)	0	0%
12	day_of_week	int	0	0%
13	time	object → datetime.time	0	0%
14	local_authority_district	int	413073	≈ 82.04%
15	local_authority_ons_district	object → string	0	0%
16	local_authority_highway	object → string	0	0%
17	local_authority_highway_current	object → string	0	0%
18	first_road_class	int	0	0%
19	first_road_number	int	0	0%
20	road_type	int	0	0%
21	speed_limit	int	15	≈ 0.003%
22	junction_detail_historic	int	59607	≈ 11.84%
23	junction_detail	int	43830	≈ 8.71%
24	junction_control	int	211708	≈ 42.05%
25	second_road_class	int	11477	≈ 2.28%
26	second_road_number	int	191038	≈ 37.94%
27	pedestrian_crossing_human_control_historic	int	63180	≈ 12.55%
28	pedestrian_crossing_physical_facilities_historic	int	63144	≈ 12.54%
29	pedestrian_crossing	int	4843	≈ 0.96%
30	light_conditions	int	19	≈ 0.004%
31	weather_conditions	int	13	≈ 0.003%
32	road_surface_conditions	int	3209	≈ 0.64%

33	special_conditions_at_site	int	63730	≈ 12.66%
34	carriageway_hazards_historic	int	63723	≈ 12.66%
35	carriageway_hazards	int	4741	≈ 0.94%
36	urban_or_rural_area	int	8	≈ 0.002%
37	did_police_officer_attend_scene_of_accident	int	1	≈0.0002%
38	trunk_road_flag	int	35920	≈ 7.13%
39	lsoa_of_accident_location	object → string	20331	≈ 4.04%
40	enhanced_severity_collision	int	241319	≈ 47.93%
41	collision_injury_based	int	0	0%
42	collision_adjusted_severity_serious	float	0	0%
43	collision_adjusted_severity_slight	float	0	0%

(d) Below are the descriptive statistics for numeric and non-numeric attributes

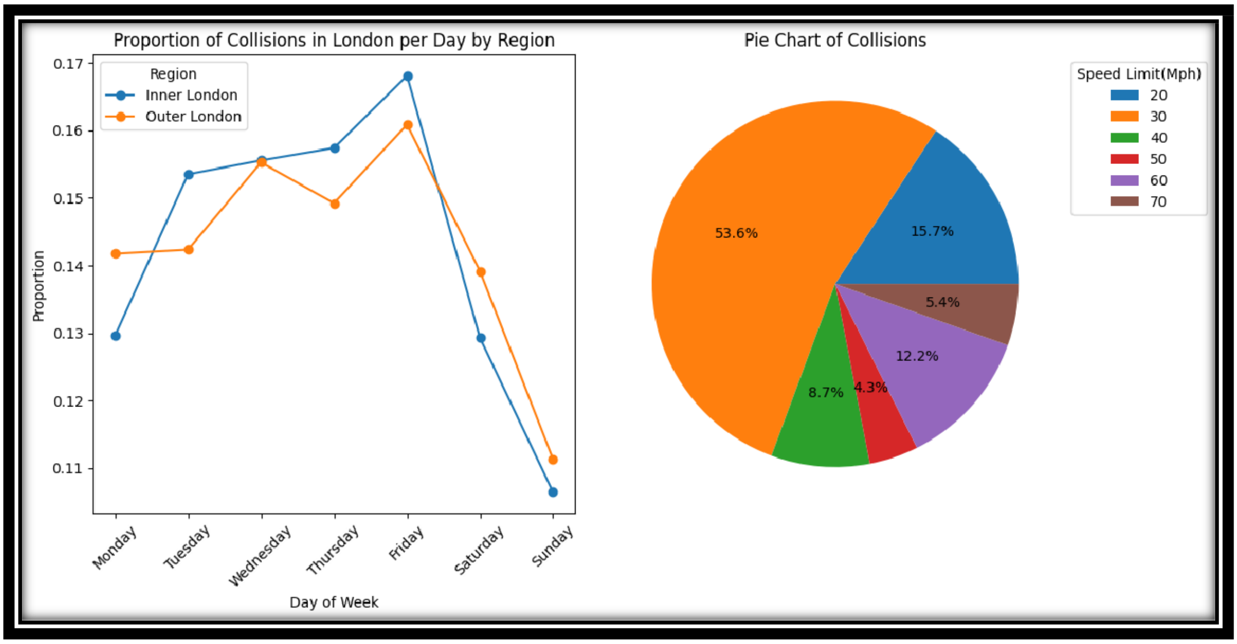
```
       collision_index  collision_year collision_ref_no  \
count           503475   503475.000000           503475
unique          503475             NaN           501768
top       2024622400537            NaN        440364708
freq                 1             NaN                3
mean               NaN     2022.044942              NaN
min                NaN     2020.000000              NaN
25%                NaN     2021.000000              NaN
50%                NaN     2022.000000              NaN
75%                NaN     2023.000000              NaN
max                NaN     2024.000000              NaN
std                NaN        1.390050              NaN

        location_easting_osgr  location_northing_osgr      longitude  \
count           503410.000000            5.034100e+05  503410.000000
unique                    NaN                     NaN            NaN
top                       NaN                     NaN            NaN
freq                      NaN                     NaN            NaN
mean            455410.085521            2.749183e+05      -1.204486
min              65947.000000            1.021100e+04      -7.497375
25%             392836.250000            1.750000e+05      -2.108384
50%             461820.500000            2.143570e+05      -1.086891
75%             529754.000000            3.818590e+05      -0.130722
max             655345.000000            1.184351e+06       1.759829
std              92797.773172            1.460445e+05       1.357086
```

```
            latitude    police_force  collision_severity  number_of_vehicles  \
count  503410.000000   503475.000000       503475.000000       503475.000000
unique           NaN             NaN                 NaN                 NaN
top              NaN             NaN                 NaN                 NaN
freq             NaN             NaN                 NaN                 NaN
mean       52.361673       27.511884            2.751807            1.828675
min        49.912210        1.000000            1.000000            1.000000
25%        51.461046        4.000000            3.000000            1.000000
50%        51.811436       22.000000            3.000000            2.000000
75%        53.328497       45.000000            3.000000            2.000000
max        60.541144       99.000000            3.000000           26.000000
std         1.315858       24.296055            0.465135            0.685515

        ...  carriageway_hazards_historic carriageway_hazards  \
count   ...                 439752.000000       498734.000000
unique  ...                           NaN                 NaN
top     ...                           NaN                 NaN
freq    ...                           NaN                 NaN
mean    ...                      0.277209            2.516875
min     ...                      0.000000            0.000000
25%     ...                      0.000000            0.000000
50%     ...                      0.000000            0.000000
75%     ...                      0.000000            0.000000
max     ...                      9.000000           99.000000
std     ...                      1.458229           13.712178
```

```
        urban_or_rural_area did_police_officer_attend_scene_of_accident  \
count         503467.000000                                503474.000000
unique                  NaN                                          NaN
top                     NaN                                          NaN
freq                    NaN                                          NaN
mean               1.324994                                     1.478434
min                1.000000                                     1.000000
25%                1.000000                                     1.000000
50%                1.000000                                     1.000000
75%                2.000000                                     2.000000
max                3.000000                                     3.000000
std                0.468628                                     0.764150

        trunk_road_flag lsoa_of_accident_location enhanced_severity_collision  \
count     467555.000000                    483144               262156.000000
unique              NaN                     35410                         NaN
top                 NaN                 E01032739                         NaN
freq                NaN                       475                         NaN
mean           1.932310                       NaN                    3.845005
min            1.000000                       NaN                    1.000000
25%            2.000000                       NaN                    3.000000
50%            2.000000                       NaN                    3.000000
75%            2.000000                       NaN                    5.000000
max            2.000000                       NaN                    7.000000
std            0.251214                       NaN                    1.590929
```

```
        collision_injury_based  collision_adjusted_severity_serious  \
count            503475.000000                        503475.000000
unique                     NaN                                  NaN
top                        NaN                                  NaN
freq                       NaN                                  NaN
mean                  0.520693                             0.236950
min                   0.000000                             0.000000
25%                   0.000000                             0.000000
50%                   1.000000                             0.000000
75%                   1.000000                             0.157065
max                   1.000000                             1.000000
std                   0.499572                             0.406133

         collision_adjusted_severity_slight
count                         503475.000000
unique                                  NaN
top                                     NaN
freq                                    NaN
mean                               0.748171
min                                0.000000
25%                                0.809562
50%                                1.000000
75%                                1.000000
max                                1.000000
std                                0.415392
```

(e) Below are 2 interesting and meaningful visualisations for the Collision dataset



This line chart shows the proportion of Collisions that happen each day of the week in Inner London compared to Outer London.

Normalising the data for each region makes it easier to highlight the pattern of when accidents are more likely to occur regardless of the total amount recorded.

If combined with 'journey purpose' data, it could reveal whether the increase in accidents during the work week is due to volume of traffic or tiredness from waking up early.

On Saturday and Sunday, as expected, Inner London is lower than Outer London as less people would be commuting in for work.

However, it would be interesting to research why Inner London on Monday has less collisions than Outer London.

This Pie Chart shows the percentage of collisions at each speed limit. This does not mean that there is an increased chance of collision at 30mph.

This is because 30mph roads make up approximately 60% of all Uk roads which is almost reflected by the pie chart.

On the other hand, this does show that one speed limit is not necessarily safer than another, therefore encouraging deeper research and understanding which will in turn provide better decision making.

Vehicle Dataset

```
fullDescribe(dfVehicle)
NUMBER OF ROWS AND COLUMN:
(920692, 32)
```

(a) In the Vehicle dataset there is 503475 Rows (Observations) and 44 Columns (Attributes/Features)

(b) (c) Table of data types and Number & percentage of missing values

Column Index	Column Name	Data Type	Missing Values	% Missing Values
0	collision_index	object → string	0	0%
1	collision_year	int	0	0%
2	collision_ref_no	object → string	0	0%
3	vehicle_reference	int	0	0%
4	vehicle_type	int	6545	≈ 0.71%
5	towing_and_articulation	int	7500	≈ 0.82%
6	vehicle_manoeuvre_historic	int	117314	≈ 12.74%
7	vehicle_manoeuvre	int	12335	≈ 1.34%
8	vehicle_direction_from	int	18680	≈ 2.03%
9	vehicle_direction_to	int	18810	≈ 2.04%
10	vehicle_location_restricted_lane_historic	int	115031	≈ 12.49%
11	vehicle_location_restricted_lane	int	10637	≈ 1.16%
12	junction_location	int	4196	≈ 0.46%
13	skidding_and_overturning	int	10093	≈ 1.10%
14	hit_object_in_carriageway	int	10022	≈ 1.09%
15	vehicle_leaving_carriageway	int	10117	≈ 1.10%
16	hit_object_off_carriageway	int	1818	≈ 0.20%
17	first_point_of_impact	int	11770	≈ 1.28%
18	vehicle_left_hand_drive	int	940	≈ 0.10%
19	journey_purpose_of_driver_historic	int	109914	≈ 11.94%
20	journey_purpose_of_driver	int	1827	≈ 0.20%
21	sex_of_driver	int	3551	≈ 0.39%
22	age_of_driver	int	137322	≈ 14.92%
23	age_band_of_driver	int	137322	≈ 14.92%
24	engine_capacity_cc	int	233745	≈ 25.39%
25	propulsion_code	int	222200	≈ 24.13%
26	age_of_vehicle	int	222433	≈ 24.16%
27	generic_make_model	object → string	241083	≈ 26.19%
28	driver_imd_decile	int	189385	≈ 20.57%
29	lsoa_of_driver	object → string	218129	≈ 23.70%
30	escooter_flag	int	0	0%
31	driver_distance_banding	int	777694	≈ 84.45%

(d) Below are the descriptive statistics for numeric and non-numeric attributes

	collision_index	collision_year	collision_ref_no	vehicle_reference
count	920692	920692.000000	920692	920692.000000
unique	503475	NaN	501768	NaN
top	2024471416027	NaN	471416027	NaN
freq	26	NaN	26	NaN
mean	NaN	2022.038721	NaN	1.552070
std	NaN	1.389831	NaN	2.564504
min	NaN	2020.000000	NaN	1.000000
25%	NaN	2021.000000	NaN	1.000000
50%	NaN	2022.000000	NaN	1.000000
75%	NaN	2023.000000	NaN	2.000000
max	NaN	2024.000000	NaN	999.000000

	vehicle_type	towing_and_articulation	vehicle_manoeuvre_historic
count	914147.000000	913192.000000	803378.000000
unique	NaN	NaN	NaN
top	NaN	NaN	NaN
freq	NaN	NaN	NaN
mean	9.919480	0.242036	20.900183
std	10.856617	1.394673	25.621954
min	1.000000	0.000000	1.000000
25%	9.000000	0.000000	9.000000
50%	9.000000	0.000000	18.000000
75%	9.000000	0.000000	18.000000
max	99.000000	9.000000	99.000000

	vehicle_manoeuvre	vehicle_direction_from	vehicle_direction_to	...
count	908357.000000	902012.000000	901882.000000	...
unique	NaN	NaN	NaN	...
top	NaN	NaN	NaN	...
freq	NaN	NaN	NaN	...
mean	20.625397	4.483473	4.528231	...
std	24.303841	2.686201	2.672091	...
min	1.000000	0.000000	0.000000	...
25%	9.000000	2.000000	2.000000	...
50%	19.000000	5.000000	5.000000	...
75%	19.000000	7.000000	7.000000	...
max	99.000000	9.000000	9.000000	...

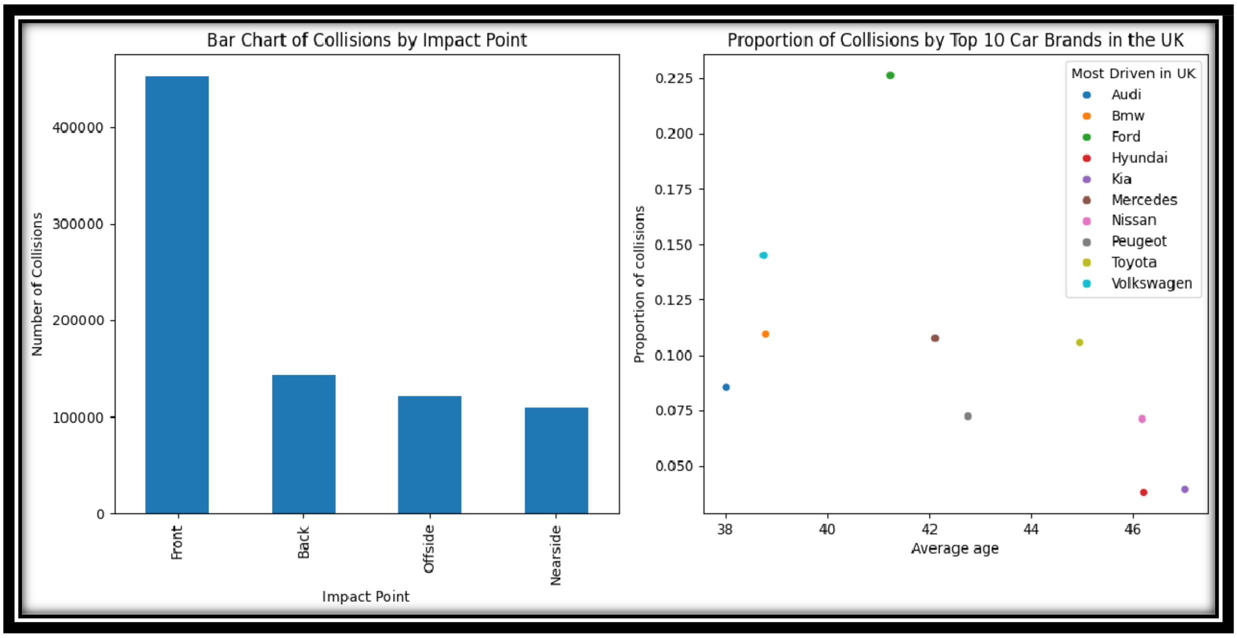
	age_of_driver	age_band_of_driver	engine_capacity_cc
count	783370.000000	783370.000000	686947.000000
unique	NaN	NaN	NaN
top	NaN	NaN	NaN
freq	NaN	NaN	NaN
mean	40.930664	6.943472	1789.692377
std	16.675895	1.808012	1573.441015
min	1.000000	1.000000	6.000000
25%	28.000000	6.000000	1240.000000
50%	38.000000	7.000000	1584.000000
75%	53.000000	8.000000	1991.000000
max	101.000000	11.000000	48000.000000

	propulsion_code	age_of_vehicle	generic_make_model
count	698492.000000	698259.000000	679609
unique	NaN	NaN	1055
top	NaN	NaN	FORD FIESTA
freq	NaN	NaN	25770
mean	1.829033	8.513897	NaN
std	1.637463	5.831241	NaN
min	1.000000	0.000000	NaN
25%	1.000000	4.000000	NaN
50%	1.000000	8.000000	NaN
75%	2.000000	13.000000	NaN
max	12.000000	123.000000	NaN

	driver_imd_decile	lsoa_of_driver	escooter_flag
count	731307.000000	702563	920692.000000
unique	NaN	36568	NaN
top	NaN	E01019456	NaN
freq	NaN	1768	NaN
mean	5.052276	NaN	0.006366
std	2.790245	NaN	0.079532
min	1.000000	NaN	0.000000
25%	3.000000	NaN	0.000000
50%	5.000000	NaN	0.000000
75%	7.000000	NaN	0.000000
max	10.000000	NaN	1.000000

	driver_distance_banding
count	142998.000000
unique	NaN
top	NaN
freq	NaN
mean	1.993657
std	1.242110
min	1.000000
25%	1.000000
50%	1.000000
75%	3.000000
max	5.000000

(e) Below are 2 interesting and meaningful visualisations for the Vehicle dataset



This column chart shows that front impacts are by far the most common, which is valuable information to insurance companies.

This suggests the most frequent claim would be front end damage meaning insurers may want to prioritize better pricing models and repair processes for those claims.

This scatter plot shows the proportion of collisions by the top 10 most driven cars in the UK against the average age of the driver involved.

There appears to be a slight negative correlation, meaning brands driven by younger drivers tend to account for a larger share of collisions.

The Ford outlier could be further investigated as it may be influenced by factors such as brand popularity, affordability, or typical usage patterns.

The plot also aligns with the general thought that brands usually associated with young drivers (Volkswagen, Audi, BMW and Ford) have a lower average driver age and a higher collision proportion suggesting that age could play a role.

1.3 Data Cleaning & Feature Engineering

(a) Data Cleaning

One data cleaning procedure will be implemented per dataset

Casualty dataset

For the Casualty Dataset, I will be using capping to help reduce the effect of outliers on "age_of_casualty" without removing the data point entirely.

I will be using 1.5x the interquartile range to decide outliers and will replace them with the nearest valid value.

```
Q1 = df["age_of_casualty"].quantile(0.25)
Q3 = df["age_of_casualty"].quantile(0.75)
IQR = Q3 - Q1

lower = Q1 - (1.5 * IQR)
upper = Q3 + (1.5 * IQR)

df["age_of_casualty"] = df["age_of_casualty"].clip(lower, upper)
```

I chose 1.5x IQR because it is the most common method used and easiest to understand.

I also considered adding jitter as there would be a pile up of ages at the boundaries but in the end chose not to.

Collision dataset

For the Collision Dataset, I will be removing columns of redundant attributes. Removing these attributes will reduce memory usage and make the dataset simpler.

The columns that will be removed are below:

Column Index	Column Name	Reason
1	collision_year	This column can be derived from "date" column.
14	local_authority_district	Approximately 82% data missing
22,27,28,34	…_historic	Only interested in current values.
40	enhance_severity_collision	Same as "severity_collision" but with less groupings and more data present.

```
historic = [column for column in df.columns if "historic" in column]
columns = historic + ["enhanced_severity_collision", "collision_year", "local_authority_district"]
df = df.drop(columns=columns)
```

Vehicle dataset

For the Vehicle Dataset, I will be filtering the data to include the top 10 most driven car brands in the UK.

The following list was decided from what was reflected in the dataset along with information from Newstrail, YouGov and RAC:

Brand
Volkswagen
Ford
BMW
Audi
Mercedes
Toyota
Nissan
Kia
Hyundai
Peugeot

```
most_popular_cars = ["volkswagen", "ford", "bmw", "audi", "mercedes",
                     "toyota","nissan", "kia", "hyundai", "peugeot"]
```

```
# Function used for mapping brand in visualisation and data cleaning

def map_brand(model):
  model = str(model).lower()
  for brand in most_popular_cars:
      if brand in model:
          return brand.capitalize()
  return np.nan
```

```
df["brand"] = df["generic_make_model"].apply(map_brand)
most_popular_cars = [brand.capitalize() for brand in most_popular_cars]
df = df[df["brand"].isin(most_popular_cars)].copy()
```

The column "generic_make_model" contains the brand but also the model. To create a new column for generic "brand" for filtering, I made the function "map_brand" which takes the list of most popular cars and returns the brand name. Using ".apply(map_brand)" on the DataFrame results in the new column which allows me to easily filter by checking whether the brand is in the list. This could also be taken further by grouping models of the same brand together.

I chose to filter by most popular car brands because they will likely make up most insurance claims. This reduces the amount of data that needs to be processed, benefiting a small company with a lack resources that would still want to benefit from cost effective predictive modelling for example.

I would want to drop "driver_distance_banding" from the dataframe as there is approximately 85% of data missing, but only one cleaning procedure is supposed to be implemented for this project and I want to display/implement different methods of data cleaning.

(b) Feature Engineering

Two feature engineering procedures will be implemented per dataset

Casualty dataset

For the Casualty Dataset, the first feature will be dropping rows where the columns have less than 3% of data missing and the second feature will be the one-hot encoding of "sex_of_casualty".

Feature 1 – Removing rows:

By removing these rows, the majority of the dataset is preserved and the quality of features are improved, creating a more consistent dataset suitable for modelling.

```
df = df.replace([-1,"-1"], np.nan)

columns = df.columns[(df.isnull().sum() / len(df) * 100) < 3].tolist()

df = df.dropna(subset=columns)
```

Feature 2 – One-hot Encoding:

The new "is_male" column is a binary indicator for the "sex_of_casualty" where 1 is male and 0 is female. This numerical format can be more useful for predictive modelling.

```
df["sex_of_casualty"] = df["sex_of_casualty"].replace(9,np.nan)
df = df.dropna(subset=["sex_of_casualty"])

df["is_male"] = (df["sex_of_casualty"] == 1).astype(int)
```

Collision dataset

For the Collision Dataset, the first feature will be the creation a new attribute "casualty_per_vehicle" derived from "number_of_causalties" and "number_of_vehicles", while the second feature will be One-hot encoding "normal_weather_conditions", derived from "light_conditions", "weather_conditions", "road_surface_conditions" and "special_conditions_at_site".

Feature 1 – Derived Attribute:

Normalising casualties by the number of vehicles allows fairer comparison across collisions of different sizes.

```
df["casualty_per_vehicle"] = df["number_of_casualties"] / df["number_of_vehicles"]
```

Feature 2 – One-hot Encoding:

The one-hot encoding can be used to split the data between normal conditions and abnormal conditions.

```
normal_conditions = (
  (df["light_conditions"] == 1) &           # Daylight
  (df["weather_conditions"] == 1) &         # Fine no high winds
  (df["road_surface_conditions"] == 1) &    # Dry
  (df["special_conditions_at_site"] == 0))  # No special conditions

df["normal_conditions"] = normal_conditions.astype(int)
```

Vehicle dataset

For the Vehicle Dataset, the first feature will be encoding the column "brand" that was created when filtering the dataset and the second feature will be handling missing values for "age_of_vehicle" through imputation.

Feature 1 – Encoding:

brand	brand_code
Audi	1
Bmw	2
Ford	3
Hyundai	4
Kia	5
Mercedes	6
Nissan	7
Peugeot	8
Toyota	9
Volkswagen	10

```
df["brand_code"] = df["brand"].astype("category").cat.codes + 1
```

This is useful for models/analysis that require numeric input as it allows access to brand information without creating multiple one-hot encoded columns. Converting the "brand" columns to categorical allows easy mapping.

Feature 2 – Imputation:

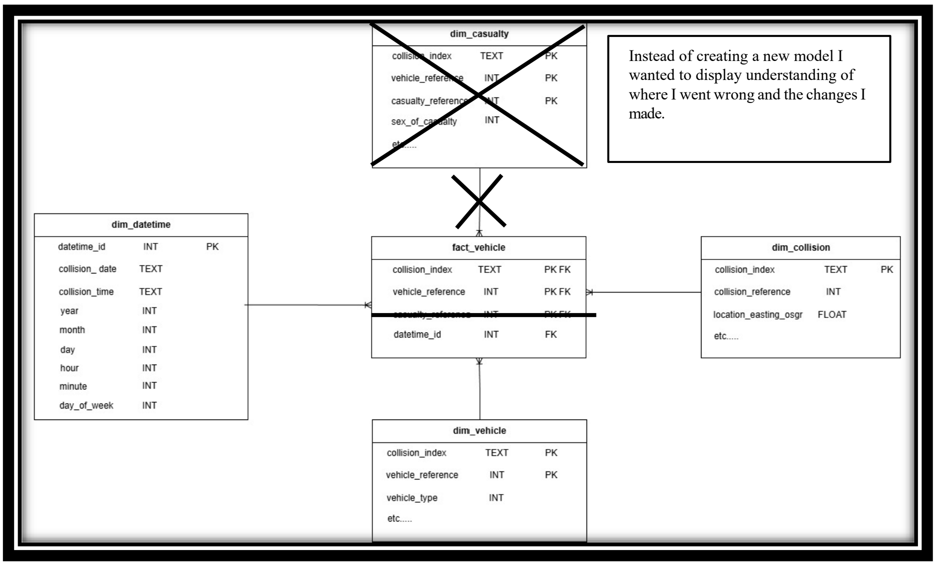
The "age of vehicle" column is extremely skewed as most cars are relatively new. Using the median provides a more realistic imputed value, resistance to extreme values and maintains the integrity of the data.

Making this feature more consistent can improve the performance of different models.

```
median = df["age_of_vehicle"].median()
df["age_of_vehicle"] = df["age_of_vehicle"].replace(-1, median).fillna(median)
```

1.4 Data Transformation

(a) Dimension-fact model



When beginning this section and deciding the fact table, I realised that the type of insurance company was not specified. This is an important fact because different insurance types would prioritise different data. For example, a life insurance company would be more concerned with casualties and their severity whereas other insurers would focus on different factors.

Since the dataset is based on UK road safety statistics, I proceeded under the assumption that car insurance was the focus however this is still an assumption. I will continue using this assumption.

This means that the fact_table should be based on the individual vehicles.

At the start of the modelling process, I struggled to identify appropriate primary and foreign keys. I decided to first check whether "collision_index" was unique.

I ran a temporary snippet of code expecting the result to be False as it would indicate that no duplicates existed and could be used as a key.

I also found in the DfT documentation (available in references), the meaning behind the columns on the dataset which supported my idea of "dim_vehicle" and "dim_casualty" requiring composite primary keys. I then realised that dim_casualty was not important to include which will be mentioned in reflection.

Therefore “dim_vehicle” will have a composite primary key of “collision_index” and vehicle_reference” as it will uniquely identify one vehicle in a collision. Based on these relationships, I designed the dimension-fact model using a star schema with the fact table “fact_vehicle” referencing the composite keys in the relevant dimensions. All data types in the Entity-Relationship Model are based on SQLite data types.

I stored date and time columns as TEXT because I encountered join issues during implementation, however these could be converted if/when required for analysis.

(b) Creating Fact table with dimensions

This section will quickly outline the process/code used to create the fact table with dimensions and may not show every bit of code for each step.

Created the database in the script directory using SQLite.

```
# Creating tables in sql using sqlite

db_name = "uk_road_safety_data.db"

# Making sure db file is created in script directory

try:
    script_dir = os.path.dirname(os.path.abspath(__file__))
except: # __file__ doesnt exist in colab
    script_dir = "."

db_path = os.path.join(script_dir, db_name)
conn = sqlite3.connect(db_path)
cursor = conn.cursor()
```

Created the tables for each dimension with the correct keys using SQL.

```
# dim_vehicle

create_dim_vehicle = '''
CREATE TABLE IF NOT EXISTS dim_vehicle(

    collision_index TEXT,
    vehicle_reference INT,
    collision_year INT,
    collision_ref_no TEXT,
    vehicle_type INT,
    towing_and_articulation INT,
    vehicle_manoeuvre_historic INT,
    vehicle_manoeuvre INT,
    vehicle_direction_from INT,
    vehicle_direction_to INT,
    vehicle_location_restricted_lane_historic INT,
    vehicle_location_restricted_lane INT,
    junction_location INT,
    skidding_and_overturning INT,
    hit_object_in_carriageway INT,
    vehicle_leaving_carriageway INT,
    hit_object_off_carriageway INT,
    first_point_of_impact INT,
    vehicle_left_hand_drive INT,
    journey_purpose_of_driver_historic INT,
    journey_purpose_of_driver INT,
    sex_of_driver INT,
    age_of_driver INT,
    age_band_of_driver INT,
    engine_capacity_cc INT,
    propulsion_code INT,
    age_of_vehicle INT,
    generic_make_model TEXT,
    driver_imd_decile INT,
    lsoa_of_driver TEXT,
    escooter_flag INT,
    driver_distance_banding INT,
    brand TEXT,
    brand_code INT,
    PRIMARY KEY (collision_index, vehicle_reference));
'''
```

For “dim_datetime”, I needed to extract the date and time features from the collision csv before creating table.

```
# dim_datetime

# Extracting the date and time and day of week from the collision table

dim_datetime = pd.DataFrame()
dim_datetime['collision_date'] = dfCollisionProcessed['date']
dim_datetime['collision_time'] = dfCollisionProcessed['time']
dim_datetime['year'] = dfCollisionProcessed['date'].apply(lambda x: x.year)
dim_datetime['month'] = dfCollisionProcessed['date'].apply(lambda x: x.month)
dim_datetime['day'] = dfCollisionProcessed['date'].apply(lambda x: x.day)
dim_datetime['day_of_week'] = dfCollisionProcessed['day_of_week']
dim_datetime['hour'] = dfCollisionProcessed['time'].apply(lambda x: x.hour)
dim_datetime['minute'] = dfCollisionProcessed['time'].apply(lambda x: x.minute)
```

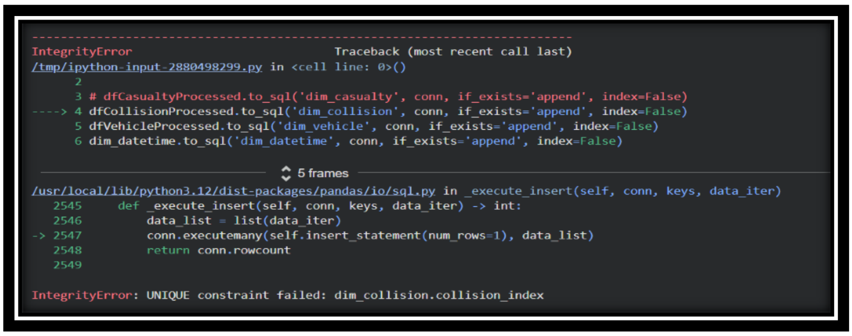
```
create_dim_datetime = '''
CREATE TABLE IF NOT EXISTS dim_datetime(

    datetime_id INTEGER PRIMARY KEY AUTOINCREMENT,
    collision_date TEXT,
    collision_time TEXT,
    year INT,
    month INT,
    day INT,
    hour INT,
    minute INT,
    day_of_week INT);
'''
```

Lastly, I created the fact table with the correct primary keys and foreign keys and then executed and committed SQL commands.

```
 # fact_vehicle

create_fact_vehicle = '''
CREATE TABLE IF NOT EXISTS fact_vehicle(
    collision_index TEXT,
    vehicle_reference INT,
    datetime_id INT,

    PRIMARY KEY (collision_index, vehicle_reference),
    FOREIGN KEY (collision_index) REFERENCES dim_collision(collision_index),
    FOREIGN KEY (collision_index, vehicle_reference) REFERENCES dim_vehicle(collision_index, vehicle_reference),
    FOREIGN KEY (datetime_id) REFERENCES dim_datetime(datetime_id)
);

'''

# Executing and committing

cursor.execute("PRAGMA foreign_keys = ON;")
#cursor.execute(create_dim_casualty) # Commented out to reflect change in ERD
cursor.execute(create_dim_collision)
cursor.execute(create_dim_vehicle)
cursor.execute(create_dim_datetime)
cursor.execute(create_fact_vehicle)
conn.commit()
```

1.5 Data Loading

The first thing I want to mention in this section is that if the database file already exists from a previous run, it should be deleted otherwise you will get a constraint error from trying to append the data twice.



```
---------------------------------------------------------------------------
IntegrityError                            Traceback (most recent call last)
/tmp/ipython-input-2880498299.py in <cell line: 0>()
      2
      3 # dfCasualtyProcessed.to_sql('dim_casualty', conn, if_exists='append', index=False)
----> 4 dfCollisionProcessed.to_sql('dim_collision', conn, if_exists='append', index=False)
      5 dfVehicleProcessed.to_sql('dim_vehicle', conn, if_exists='append', index=False)
      6 dim_datetime.to_sql('dim_datetime', conn, if_exists='append', index=False)

5 frames
/usr/local/lib/python3.12/dist-packages/pandas/io/sql.py in _execute_insert(self, conn, keys, data_iter)
   2545     def _execute_insert(self, conn, keys, data_iter) -> int:
   2546         data_list = list(data_iter)
-> 2547         conn.executemany(self.insert_statement(num_rows=1), data_list)
   2548         return conn.rowcount
   2549

IntegrityError: UNIQUE constraint failed: dim_collision.collision_index
```

First, I load the dimensions because "fact_vehicle" is dependent on them.

```
# Loading data into database

# dfCasualtyProcessed.to_sql('dim_casualty', conn, if_exists='append', index=False) # Commented out to reflect change in ERD
dfCollisionProcessed.to_sql('dim_collision', conn, if_exists='append', index=False)
dfVehicleProcessed.to_sql('dim_vehicle', conn, if_exists='append', index=False)
dim_datetime.to_sql('dim_datetime', conn, if_exists='append', index=False)
```

Then I merge on the correct keys using the correct joins.

I did encounter a problem when trying to merge "dim_datetime". I added and changed the code a lot by reacting to console errors which made me lose track of what was actually happening.

I went from getting errors to the table finally being created but the time_id column in the fact table was actually empty.

```
# Merging Collision

fact_vehicle = dfVehicleProcessed.merge(
    dfCollisionProcessed[['collision_index', 'date', 'time']],
    on='collision_index',
    how='left'
)

# Merging datetime

dim_datetime_db = pd.read_sql_query("SELECT * FROM dim_datetime", conn) # Need to get autoincremented datetime_id


# Ensure type consistency for merging date/time

fact_vehicle['date_str'] = pd.to_datetime(fact_vehicle['date']).dt.date.astype(str)
fact_vehicle['time_str'] = pd.to_datetime(fact_vehicle['time'], format='%H:%M:%S', errors='coerce').dt.time.astype(str)

dim_datetime_db['collision_date_str'] = pd.to_datetime(dim_datetime_db['collision_date']).dt.date.astype(str)
dim_datetime_db['collision_time_str'] = pd.to_datetime(dim_datetime_db['collision_time']).dt.time.astype(str)

# Merging to get datetime_id (changed to inner to ensure datetime_id is not null)
fact_vehicle = fact_vehicle.merge(
    dim_datetime_db[['datetime_id', 'collision_date_str', 'collision_time_str']],
    left_on=['date_str', 'time_str'],
    right_on=['collision_date_str', 'collision_time_str'],
    how='inner'
)

# Keeping columns needed for fact table
```

In the end the plan was to convert the date and time to strings so I added different lines of code until the time_id was populated in the fact table. In the end I still don't understand why it worked. I consider this partially a fail from me as I don't understand how I got it semi working and is based on luck. In a way I "frankensteined" this section by trying to replicate results from my previous python projects, hoping it would work.

Finally, I kept only the columns needed for the fact table. I automatically used “if_exists replace” which removed the constraints I created so I changed it to append. This was the moment I realized the key constraints were not being enforced. When I tried to enforce them, I would get an error which in the end was caused by “dim_casualty” where I then also realized that for the purpose of a car insurance company, this data is not necessarily needed. (assumption).

Before with “dim_casualty”:

```
# Keeping columns needed for fact table

fact_vehicle = fact_vehicle[['collision_index', 'vehicle_reference', 'casualty_reference', 'datetime_id']]

fact_vehicle.to_sql('fact_vehicle', conn, if_exists='replace', index=False) # For some reason only works with replace

conn.close() # Close connection to db
```

After “removing dim_casualty”:

```
# Keeping columns needed for fact table

fact_vehicle = fact_vehicle[['collision_index', 'vehicle_reference', 'datetime_id']]
fact_vehicle = fact_vehicle.drop_duplicates(subset=['collision_index', 'vehicle_reference']) # Duplicates exist from merging
fact_vehicle.to_sql('fact_vehicle', conn, if_exists='append', index=False)

conn.close() # Close connection after all operations are done
```

After loading the data into the database I closed the connection. The database file will be in the same directory as the script and is named “uk_road_safety_data”

1.6 Data Pipeline Testing

In this section I create 3 simple tests for each stage of the pipeline. “pipelineTest1” represents the Extract, “pipelineTest2” represents the Transform and “pipelinetest3” represents the Load in the ETL Pipeline model diagram shown in the introduction.

(a) Data loader test

The first test checks that the correct number of columns are present.

This is the result:

```
Test 1 Data Loading
PASS
PASS
PASS
Test 1 Complete
```

```python
def pipelineTest1(dfCasualty, dfCollision, dfVehicle):

 test_list = []

 print("Test 1 Data Loading")

 # A Pass if the csv has the correct number of columns

 if dfCasualty.shape[1] == 23:
     print("PASS")
     test_list.append("PASS")
 else:
     print(f"FAIL: Expected 23 columns in Casualty, got {dfCasualty.shape[1]}")
     test_list.append("FAIL")

 if dfCollision.shape[1] == 44:
     print("PASS")
     test_list.append("PASS")
 else:
     print(f"FAIL: Expected 44 columns in Collision, got {dfCollision.shape[1]}")
     test_list.append("FAIL")

 if dfVehicle.shape[1] == 32:
     print("PASS")
     test_list.append("PASS")
 else:
     print(f"FAIL: Expected 32 columns in Vehicle, got {dfVehicle.shape[1]}")
     test_list.append("FAIL")

 print("Test 1 Complete")  #

 return test_list
```

(b) Transformation testing

The second test checks if any of the age columns are negative.

This is the result:

```
Test 2 Data Transforming
PASS
PASS
PASS
Test 2 Complete
```

```python
def pipelineTest2(dfCasualtyProcessed, dfCollisionProcessed, dfVehicleProcessed, test_list = None):

 if test_list is None:
     test_list = []

 print("Test 2 Data Transforming")

 # A Fail if any of the age columns are negative

 if (dfCasualtyProcessed["age_of_casualty"]<0).any():
   print("FAIL: Negative age_of_casualty in Casualty")
   test_list.append("FAIL")
 else:
   print("PASS")
   test_list.append("PASS")

 if (dfVehicleProcessed["age_of_driver"]<0).any():
   print("FAIL: Negative age_of_driver in Vehicle")
   test_list.append("FAIL")
 else:
   print("PASS")
   test_list.append("PASS")

 if(dfVehicleProcessed["age_of_vehicle"]<0).any():
   print("FAIL: Negative age_of_vehicle in Vehicle")
   test_list.append("FAIL")
 else:
   print("PASS")
   test_list.append("PASS")

 print("Test 2 Complete")


 return test_list
```

(c) Serving/loading tests

This final test checks if the data has been loaded correctly into each table by checking if a row of data is returned from an SQL query.

This is the result along with all test results:

```python
# Pipeline Testing

# Each test function prints result of each test but also returns a list

test_list = pipelineTest1(dfCasualty,dfCollision,dfVehicle) # Function for testing csv loading
test_list = pipelineTest2(dfCasualtyProcessed,dfCollisionProcessed,dfVehicleProcessed, test_list) # Function for testing data tran
test_list = pipelineTest3(test_list) # Function for testing db file
print("\nFull Test Results: ")
print(test_list)


if "FAIL" in test_list:
    print("One or more tests failed")
else:
    print("All Pipeline Tests Passed")
```

```
Test 3 Data Serving
PASS
PASS
PASS
PASS
Test 3 Complete

Full Test Results:
['PASS', 'PASS', 'PASS', 'PASS', 'PASS', 'PASS', 'PASS', 'PASS', 'PASS', 'PASS']
All Pipeline Tests Passed
```

```python
def pipelineTest3(test_list = None):

  if test_list is None:
    test_list = []

  print("Test 3 Data Serving")

  # A Pass if each table has at least one row

  db_name = "uk_road_safety_data.db"

  # Making directory is where script is running

  try:
    script_dir = os.path.dirname(os.path.abspath(__file__))
  except:
    script_dir = "."

  db_path = os.path.join(script_dir, db_name)

  conn = sqlite3.connect(db_path)
  cursor = conn.cursor()


  tables = ["fact_vehicle", "dim_collision", "dim_vehicle", "dim_datetime"] # Each table name in db

  for table in tables:
    cursor.execute(f"SELECT 1 FROM {table} LIMIT 1;")  # Check if theres any data
    result = cursor.fetchone() # One row of Data
    if result is None:

      print(f"FAIL: {table} has no data")
      test_list.append("FAIL")

    else:
      print("PASS")
      test_list.append("PASS")

  print("Test 3 Complete")

  conn.close()

  return test_list
```

Chapter 2: Reflection <2000 Words

2.1 What went wrong?

- The further into the script I got, the more bored and sloppier my code and documentation ended up and I started to develop an attitude of "as long as it works".
- I have been coding in a lot of Java so at times I use camel case naming. For example, "dfCasualty" instead of "df_casualty". I also don't follow other standard python coding practices.
- I don't fully understand fact tables and entity relationship diagrams. I found self-study tricky for this section.
- I may or may not have got the relationships wrong in my fact-dimension model and misunderstood what a fact-dimension model even is.
- Sometimes I change ideas midway and leave some code in, either because it breaks something or because my code still works with it in.
- Usually, I use Astah for creating diagrams, however ERDs are not available on the student version so I struggled with online diagram tools which were all bad in my opinion.

2.2 What went right?

- I saw the error I made where I had replaced the fact table I made using “replace” instead of “append”, resulting in the removal of the key constraints I made. This led me down the path changing the model.

- I used functions to make my code easier to manage and change.

- I challenged myself to implement ideas even when I don’t know where to start, however I only kept in if I fully understood it. (Apart from one instance)

- Tested my code in multiple environments such as in Google Colab, Visual Studio Code, Python IDLE and just running the file itself, which helped me identify certain problems and allowed for greater compatibility.

2.3 What did I learn?

- Originally my pipeline test was failing, but it turns out it was due to the mapping of the brand in the vehicle csv. I didn't use df.copy() causing the df to be edited outside the function which resulted in 33 columns instead of 32. So now I understand the importance of using df.copy().

- I learned that I can actually pass a function into .apply() and also use lambda which were both very helpful.

- My pipeline tests are using if and else statements where I later found out I could use "assert" instead.

2.4 What would I do differently?

- I would do more error handling as I just assumed everything would be inputted correctly for simplicity.
- I would have more robust pipeline tests, for example I am only fetching one row for efficiency sake and not actually verifying the data in it.
- I would try to cut down the lines of code and make it more efficient.
- I would probably make the severity labels in the visualisation ordered.
- A nicer formatted documentation.
- I saw that it was possible to skip rows of data that are already present when appending by ignoring it instead of throwing an error, but I didn't want to use code I didn't fully understand. That's why I opted for not deleting the database every time the script is ran, and mentioned that the db should be deleted if ran again, as I may try implement that in the future in my own time.
- I would have columns dropped automatically based on percentage of data missing.
- Automatically check the data type and convert if needed, instead of manually making a list of columns names.

Reflection Final Word Count – ≈ 635

I would say that most of my reflection is displayed throughout the documentation in different sections.

References

Canva [no date]

Page cover inspiration [online]

Available at: https://www.canva.com/templates/EAFBSZgGLS8-minimalist-simple-annual-report/

[Accessed 26 October 2025]

Department for Transport (DfT) [2025]

Road Safety Data [online]

Available at: https://www.gov.uk/government/statistics/road-safety-data

[Accessed 26 October 2025]

Road Open Data Guide [online]

Available at: https://www.gov.uk/government/statistics/road-safety-data

[Accessed 27 October 2025]

SQLPey [2025]

Python Get Current Script Directory Safely [online]

Available at: https://sqlpey.com/python/python-get-script-directory/

[Accessed 27 October 2025]

Draw.io [2025]

Online Diagram Software [online]

Available at: https://app.diagrams.net/

[Accessed 30 October 2025]

Newstrail [2025]

Popular car brands in the UK in 2025 [online]

Available at: https://www.newstrail.com/popular-car-brands-in-the-uk-in-2025/

[Accessed 4 November 2025]

YouGov [2025]

The most popular car brands (Q3 2025) [online]

Available at: https://yougov.co.uk/ratings/travel/popularity/car-brands/all

[Accessed 4 November 2025]

RAC [2025]

The top 10 most popular cars in the UK [online]

Available at: https://www.rac.co.uk/drive/advice/buying-and-selling-guides/the-top-10-most-popular-cars-in-the-uk/

[Accessed 4 November 2025]

Pandas [no date]

pandas.to_datetime [online]

Available at: https://sqlpey.com/python/python-get-script-directory/

[Accessed 10 November 2025]

Logos-World [2021]

Python Emblem [online] - The Python logo is a trademark of the Python Software Foundation (PSF).

Available at: https://logos-world.net/wp-content/uploads/2021/10/Python-Emblem.png

[Accessed 17 November 2025]

Stack Overflow [no date]

How does plt.gca work internally [online]

Available at: https://stackoverflow.com/questions/45381589/how-does-plt-gca-work-internally

[Accessed 4 November 2025]

Different results with observed=True/False in pandas groupby [online]

Available at: https://stackoverflow.com/questions/56339040/different-results-with-observed-true-false-in-pandas-groupby

[Accessed 5 November 2025]

Sqlite / SQLAlchemy: how to enforce Foreign Keys? [online]

Available at: https://stackoverflow.com/questions/2614984/sqlite-sqlalchemy-how-to-enforce-foreign-keys

[Accessed 12 November 2025]

pandas to_datetime throwing a value error for incorrect date format [online]

Available at: https://stackoverflow.com/questions/69260096/pandas-to-datetime-throwing-a-value-error-for-incorrect-date-format

[Accessed 13 November 2025]

pandas extract year from datetime: df['year'] = df['date'].year is not working [online]

Available at: https://stackoverflow.com/questions/30405413/pandas-extract-year-from-datetime-dfyear-dfdate-year-is-not-working

[Accessed 13 November 2025]