INTRODUCTION

In this work, I'm analysing the fatal traffic accidents that occurred in Great Britain in the year 2020. I need to access the following external links in order to understand the meaning of each column in the reported log, which includes both major and non-fatal traffic accidents: stats20-2011.pdf, Reported road casualties in Great Britain: notes, definitions, symbols and conventions, and Road Traffic Accidents Statistics Form. The logged data used was also stored in SQLite database through the link accident_data_v1.0.0_2023.db.

Data Analysis

1. I used the SQL syntax to filtered out the day, day of the week, the time which is the hour at which the accident happened and the number of accidents that happened for does days.

SQL 1

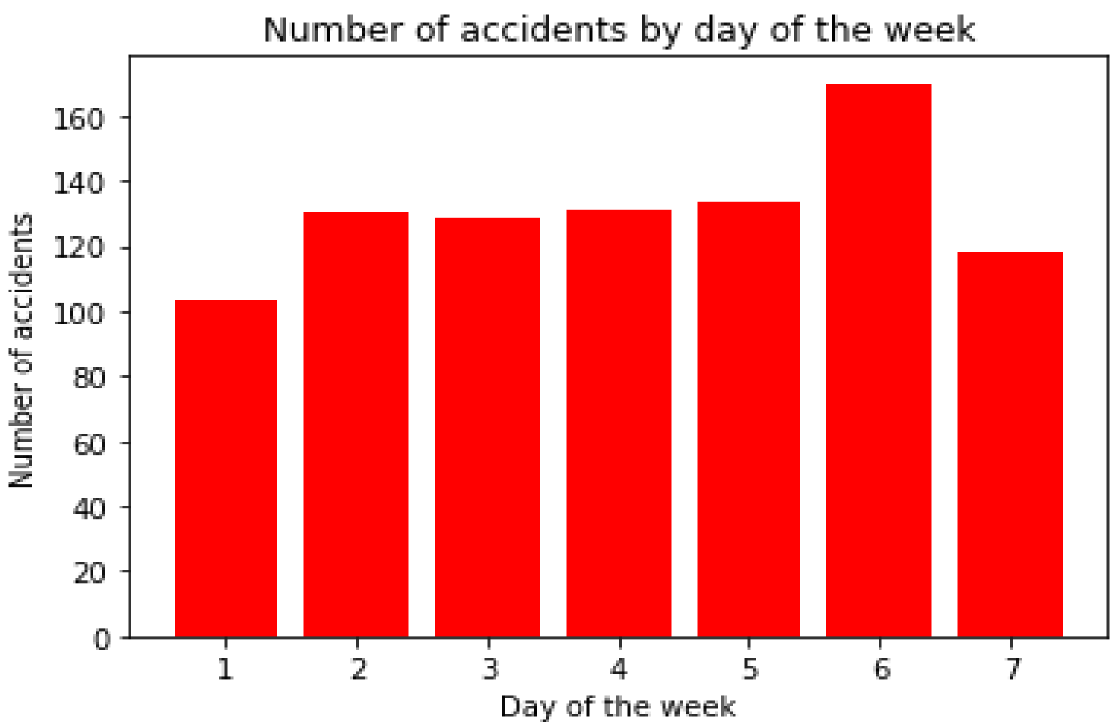
```sql

SELECT day_of_week,time, COUNT(*) AS 'Number of Accidents'
FROM accident
WHERE accident_year = 2020
GROUP BY day_of_week, time
ORDER BY COUNT(*) DESC;
```

	day_of_week	time	Number of Accidents
1	6	17:00	170
2	5	18:00	134
3	6	15:00	132
4	4	17:00	131
5	2	16:00	130
6	3	17:00	129
7	5	15:00	128
8	5	16:00	127
9	6	16:00	126
10	3	18:00	124

The 6th day of the week has the highest occurrence of accident and this accident occurs at 17:00.

I had to export the result directly to jupyter notebook for data visualisation and analysis.

```python
# creating a bar plot
plt.bar(data["day_of_week"], data["Number of Accidents"], color = 'r')

#Adding a title
plt.title("Number of accidents by day of the week")

# Adding labels to the axes
plt.xlabel("Day of the week")
plt.ylabel("Number of accidents")

#showing the plot
plt.show()
```



The figure above depicts a bar plot of the number of accidents by weekday. The fifth and sixth days of the week had the most accidents, while the first day of the week had the fewest.

For Significant Hours Accidents occur:.

```sql
SELECT accident_year,time,date,count(*) AS Number_of_accidents
from accident
where accident_year=2020
group by time
ORDER by Number_of_Accidents DESC
LIMIT 30;
```

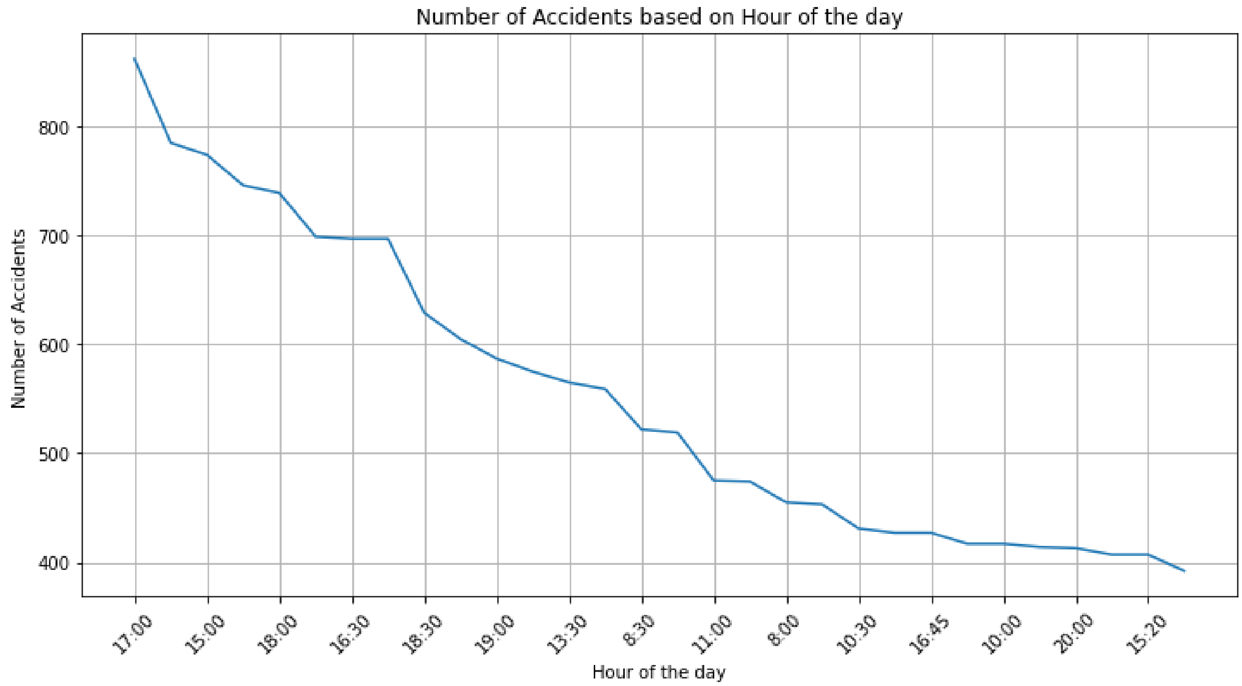
The visualisation was a time series that examined the trends of the accidents at various time intervals. The procedures are listed below.

```python
In [4]: hour = pd.read_csv(r"C:\Users\hp pc\Desktop\Quincy\accident\question 1b.csv")
        hour.head()
```

Out[4]:

	accident_year	time	date	Number_of_accidents
0	2020	17:00	4/1/2020	862
1	2020	16:00	6/1/2020	785
2	2020	15:00	1/1/2020	774
3	2020	17:30	6/1/2020	746
4	2020	18:00	3/1/2020	739

```python
In [5]: # Increase the size of the chart
        plt.figure(figsize=(12, 6))

        # Plotting the time series
        plt.plot(hour.time, hour.Number_of_accidents)

        # Adding title
        plt.title("Number of Accidents based on Hour of the day")

        # Adding labels to the axes
        plt.xlabel("Hour of the day")
        plt.ylabel("Number of Accidents")

        # Show every 2nd x-axis label and rotate them
        plt.xticks(hour.time[::2], rotation=45)
```



According to the visualisation, the majority of the incidents occurred in the evening about 17:00, which is presumably a period when people are leaving work or enterprises.

2. Motorcycle 125cc and Under.

a. Day of the Week Analysis

```
SELECT vehicle.vehicle_type, accident.accident_year, accident.day_of_week, accident.time, accident.date, COUNT(accident.accident_index) AS "Number of Accidents"
FROM accident
JOIN vehicle ON accident.accident_index = vehicle.accident_index
WHERE accident.accident_year = 2020 AND (vehicle.vehicle_type = 2 OR vehicle.vehicle_type = 3)
GROUP BY accident.day_of_week
ORDER BY "Number of Accidents" DESC;
```

	vehicle_type	accident_year	day_of_week	time	date	Number of Accidents
1	3	2020	1	13:50	05/01/2020	948
2	3	2020	2	00:44	06/01/2020	1173
3	3	2020	3	13:00	07/01/2020	1202
4	3	2020	4	03:25	01/01/2020	1249
5	3	2020	5	13:20	02/01/2020	1389
6	3	2020	6	07:50	03/01/2020	1474
7	2	2020	7	20:25	04/01/2020	1216

I selected some columns from the accident table which contains information about all the accidents that have occurred, and the vehicle table which contains information about the vehicles that were involved in the accidents.

I first join the accident and vehicle tables on the accident index column. The code then uses the WHERE clause to filter the results to only include accidents where the vehicle type is 2 or 3 and where the year is 2020.

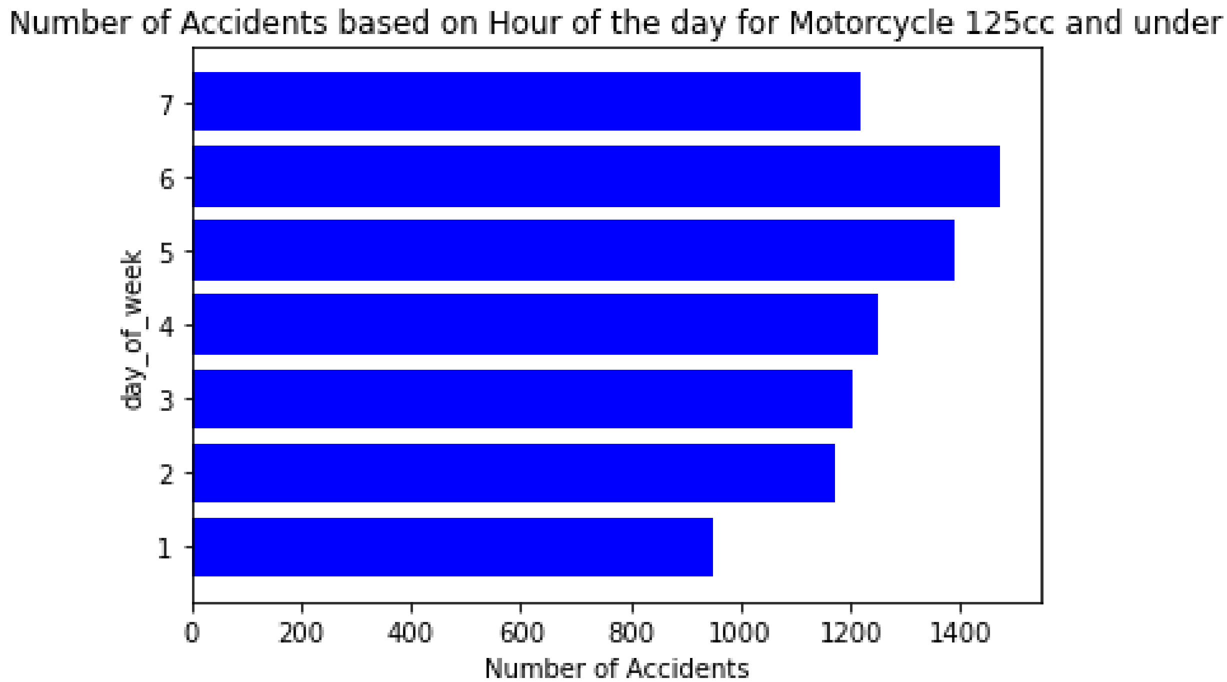
b. Significant Day_of_week Analysis

In [6]:
```
motor_125 = pd.read_csv(r"C:\Users\hp pc\Desktop\Quincy\accident\question 2a.csv")
motor_125.head()
```

Out[6]:

	vehicle_type	accident_year	day_of_week	time	date	Number_of_Accidents
0	3	2020	6	7:50	3/1/2020	1474
1	3	2020	5	13:20	2/1/2020	1389
2	3	2020	4	3:25	1/1/2020	1249
3	2	2020	7	20:25	4/1/2020	1216
4	3	2020	3	13:00	7/1/2020	1202

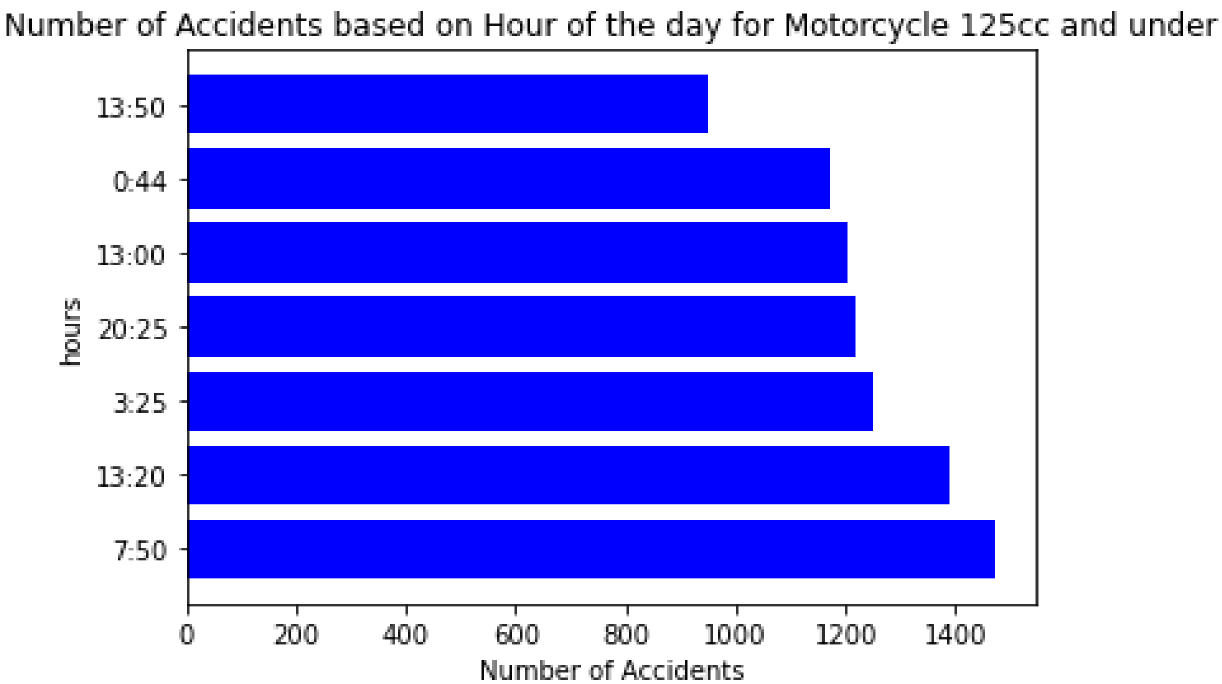
In [7]:
```
# creating a bar plot
plt.barh(motor_125.day_of_week, motor_125.Number_of_Accidents, color = 'b')

#Adding a title
plt.title("Number of Accidents based on day of the week for Motorcycle 125cc and under")

# Adding labels to the axes
plt.xlabel("Number of Accidents")
plt.ylabel("day_of_week")

# Showing the plot
plt.show()
```



The code's output is depicted in the figure above. According to the graph, the majority of the incidents occurred on Saturday, followed by Friday and Thursday, with the first day of the week having the fewest documented accidents.

c. Significant Hours Analysis

```
# creating a bar plot
plt.barh(motor_125.time, motor_125.Number_of_Accidents, color = 'b')

#Adding a title
plt.title("Number of Accidents based on Hour of the day for Motorcycle 125cc and under")

# Adding labels to the axes
plt.xlabel("Number of Accidents")
plt.ylabel("hours")

# Showing the plot
plt.show()
```



I sorted the data by time and only included the hours with the highest number of accidents.

This demonstrates that 7:50 is one of the most dangerous times of the day.

2.2. Motorcycle over 125cc and up to 500cc

```sql
SELECT vehicle.vehicle_type, accident.accident_year, accident.day_of_week, accident.time, accident.date, COUNT(accident.accident_index) AS "Number of Accidents"
FROM accident
JOIN vehicle ON accident.accident_index = vehicle.accident_index
WHERE accident.accident_year = 2020 AND (vehicle.vehicle_type = 4)
GROUP BY accident.day_of_week
ORDER BY "Number of Accidents" DESC;
```

	vehicle_type	accident_year	day_of_week	time	date	Number of Accidents
1	4	2020	6	14:35	10/01/2020	267
2	4	2020	5	14:17	02/01/2020	257
3	4	2020	4	14:55	15/01/2020	251
4	4	2020	7	00:27	11/01/2020	237
5	4	2020	3	16:33	07/01/2020	234
6	4	2020	2	15:45	13/01/2020	227
7	4	2020	1	19:05	12/01/2020	218

a. Day of the Week Analysis

In [9]:

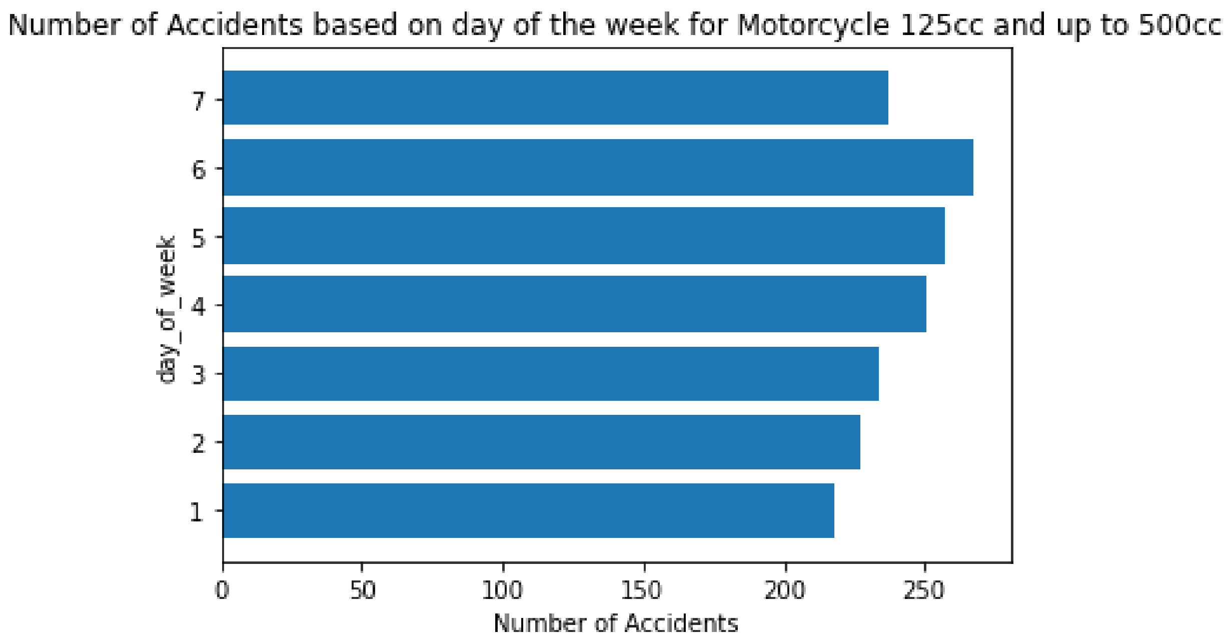
```python
motor_125_over = pd.read_csv(r"C:\Users\hp pc\Desktop\Quincy\accident\question 2b.csv")
motor_125_over.head()
```

Out[9]:

	vehicle_type	accident_year	day_of_week	time	date	Number_of_Accidents
0	4	2020	6	14:35	10/1/2020	267
1	4	2020	5	14:17	2/1/2020	257
2	4	2020	4	14:55	15/01/2020	251
3	4	2020	7	0:27	11/1/2020	237
4	4	2020	3	16:33	7/1/2020	234

In [10]:

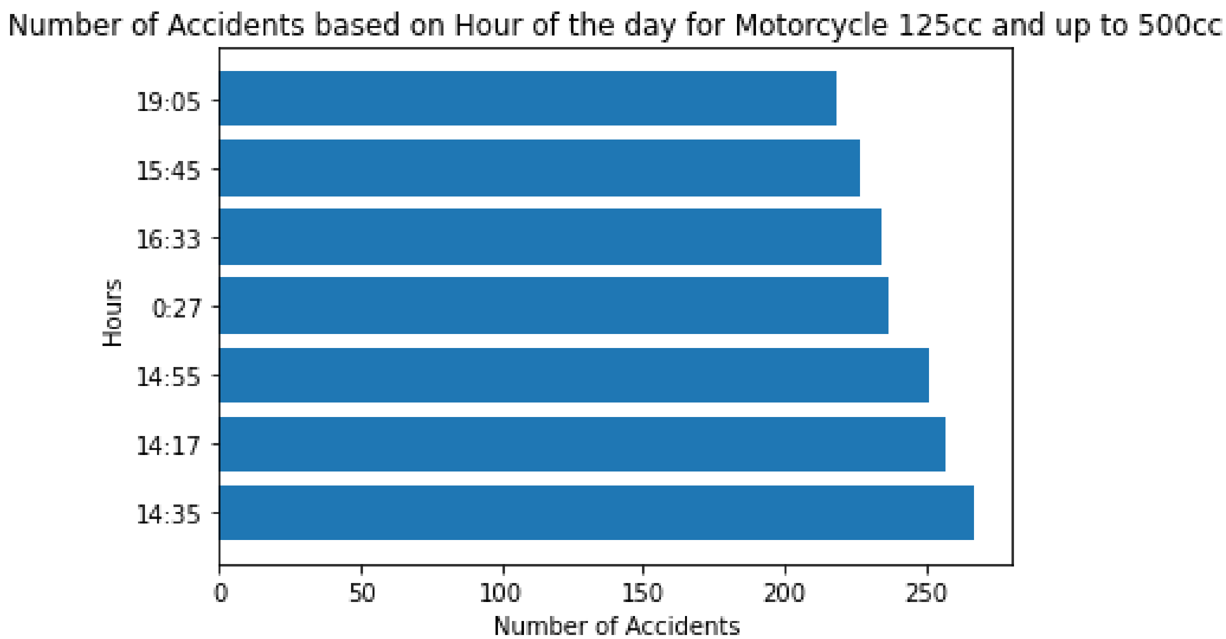
```python
# creating a bar plot
plt.barh(motor_125_over.day_of_week, motor_125_over.Number_of_Accidents)

#Adding a title
plt.title("Number of Accidents based on day of the week for Motorcycle 125cc and up to 500cc")

# Adding labels to the axes
plt.xlabel("Number of Accidents")
plt.ylabel("day_of_week")

# Showing the plot
plt.show()
```



The day with the most accidents was Saturday, followed by Friday and Thursday. The first day of the week has the fewest documented accidents for motorcycles larger than 125cc and up to 500cc.

b. Significant Hours Analysis

```
# creating a bar plot
plt.barh(motor_125_over.time, motor_125_over.Number_of_Accidents)

#Adding a title
plt.title("Number of Accidents based on Hour of the day for Motorcycle 125cc and up to 500cc")

# Adding labels to the axes
plt.xlabel("Number of Accidents")
plt.ylabel("Hours")

# Showing the plot
plt.show()
```



The motorcycles over 500cc, accidents occurred mostly from 14:00 to 14:55. Where 14:35 is the time with the highest number of accidents occurred.

2.3. Motorcycle over 500cc

```sql
SELECT vehicle.vehicle_type, accident.accident_year, accident.day_of_week, accident.time, accident.date, COUNT(accident.accident_index) AS "Number of Accidents"
FROM accident
JOIN vehicle ON accident.accident_index = vehicle.accident_index
WHERE accident.accident_year = 2020 AND (vehicle.vehicle_type = 5)
GROUP BY accident.day_of_week
ORDER BY "Number of Accidents" DESC;
```

	vehicle_type	accident_year	day_of_week	time	date	Number of Accidents
1	5	2020	1	16:35	05/01/2020	675
2	5	2020	6	19:50	03/01/2020	567
3	5	2020	7	13:00	04/01/2020	550
4	5	2020	5	15:24	02/01/2020	526
5	5	2020	4	15:39	08/01/2020	501
6	5	2020	3	00:50	07/01/2020	497
7	5	2020	2	08:34	06/01/2020	468

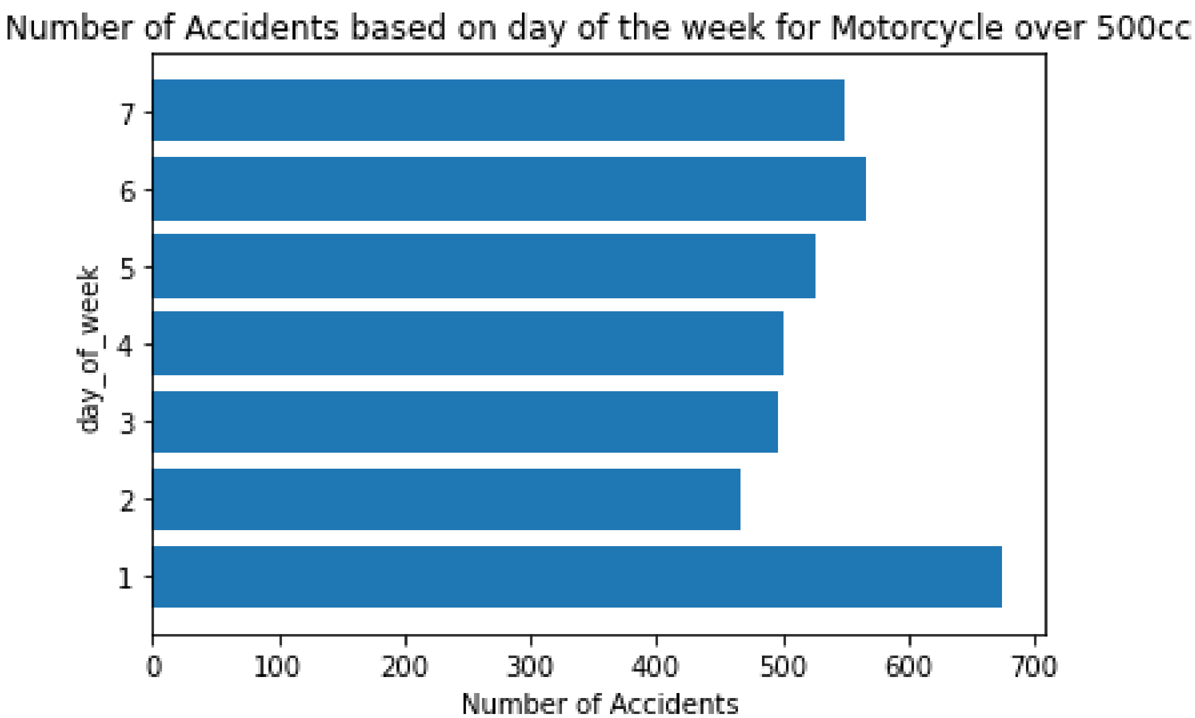
a. Day of the Week Analysis

In [12]:
```
motor_500_over = pd.read_csv(r"C:\Users\hp pc\Desktop\Quincy\accident\question 2c.csv")
motor_500_over.head()
```

Out[12]:

	vehicle_type	accident_year	day_of_week	time	date	Number_of_Accidents
0	5	2020	1	16:35	5/1/2020	675
1	5	2020	6	19:50	3/1/2020	567
2	5	2020	7	13:00	4/1/2020	550
3	5	2020	5	15:24	2/1/2020	526
4	5	2020	4	15:39	8/1/2020	501

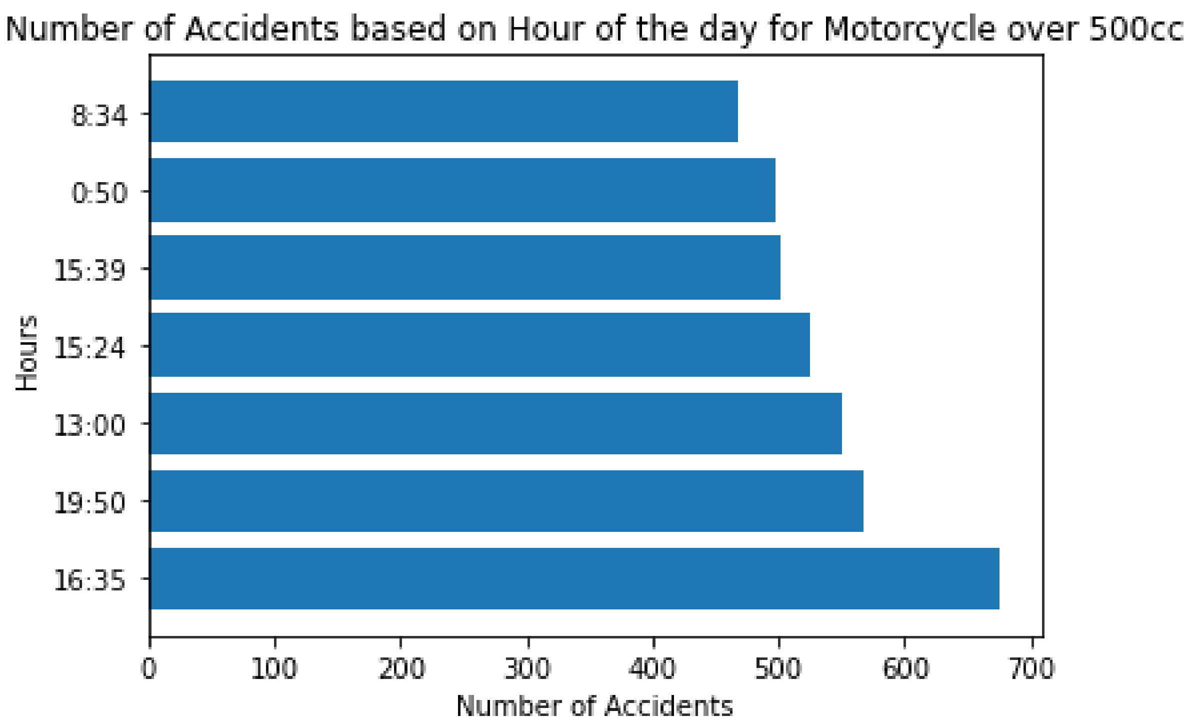
In [13]:
```
# creating a bar plot
plt.barh(motor_500_over.day_of_week, motor_500_over.Number_of_Accidents)

#Adding a title
plt.title("Number of Accidents based on day of the week for Motorcycle over 500cc")

# Adding labels to the axes
plt.xlabel("Number of Accidents")
plt.ylabel("day_of_week")

# Showing the plot
plt.show()
```



According to this table, the first day of the week has the largest number of accidents for motorbikes above 500cc, followed by Saturday, the sixth day, and the second day of the week, which have the lowest number of accidents.

b. Significant Hours Analysis

```
# creating a bar plot
plt.barh(motor_500_over.time, motor_500_over.Number_of_Accidents)

#Adding a title
plt.title("Number of Accidents based on Hour of the day for Motorcycle over 500cc")

# Adding labels to the axes
plt.xlabel("Number of Accidents")
plt.ylabel("Hours")

# Showing the plot
plt.show()
```



The table above illustrates the hours of the day with the highest number of accidents for bikes above 500cc, with accidents usually happening around 16:35 and 8:34 having the lowest accident occurrence.

3. I used the SQL syntax to filtered day of the week, time which is the hour at which the accident happened and the number of accidents that happened for does days where (casualty class = 3) meaning pedestrians.

```sql
SELECT accident.accident_year, accident.day_of_week,
 accident.time, COUNT(accident.accident_index) AS "Number of Accidents"
FROM accident
JOIN casualty ON accident.accident_index = casualty.accident_index
WHERE accident.accident_year = 2020 AND (casualty.casualty_class = 3)
GROUP BY accident.day_of_week
ORDER BY "Number of Accidents" DESC;
```

	accident_year	day_of_week	time	Number of Accidents
1	2020	6	07:22	2543
2	2020	5	09:35	2366
3	2020	3	09:00	2267
4	2020	4	01:25	2247
5	2020	2	13:55	2207
6	2020	7	20:25	1878
7	2020	1	06:48	1242

a. **Day of the Week Analysis**

5]:

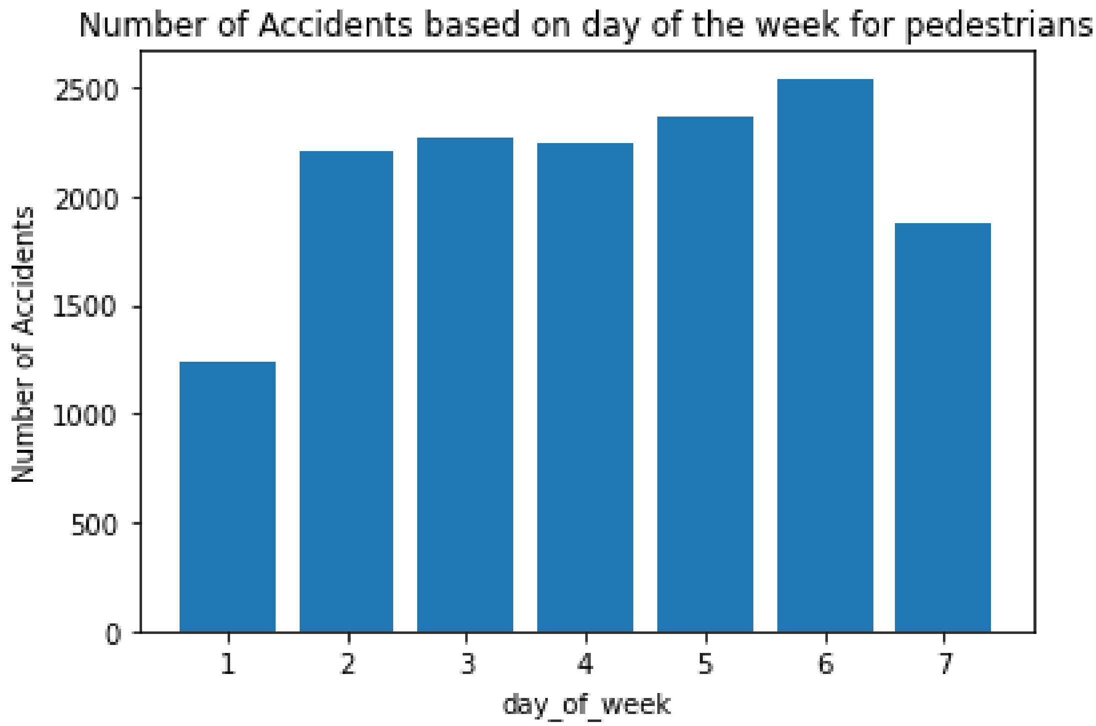
```python
pedestrians = pd.read_csv(r"C:\Users\hp pc\Desktop\Quincy\accident\question 3.csv")
pedestrians.head()
```

5]:

	accident_year	day_of_week	time	Number_of_Accidents
0	2020	6	7:22	2543
1	2020	5	9:35	2366
2	2020	3	9:00	2267
3	2020	4	1:25	2247
4	2020	2	13:55	2207

6]:

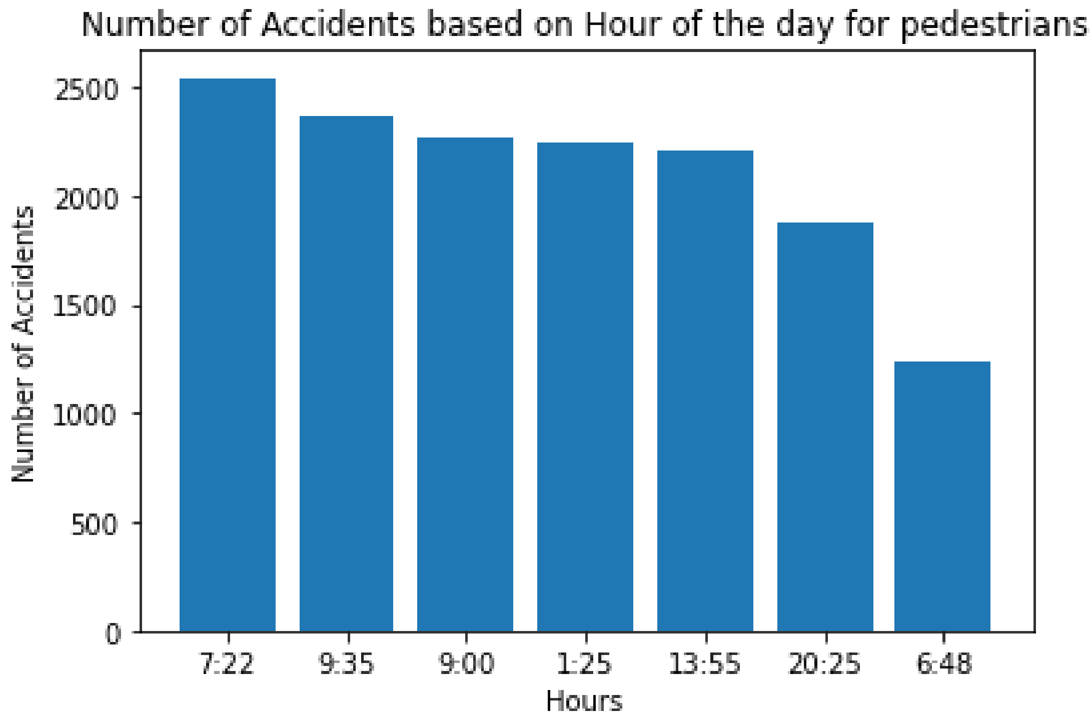
```python
# creating a bar plot
plt.bar(pedestrians.day_of_week, pedestrians.Number_of_Accidents)

#Adding a title
plt.title("Number of Accidents based on day of the week for pedestrians")

# Adding labels to the axes
plt.xlabel("day_of_week")
plt.ylabel("Number of Accidents")

# Showing the plot
plt.show()
```



The chart above shows that the most accidents involving pedestrians' accident occurs on Saturday at 6th day the week with total of 2543 accidents on that day. Monday which is 1st day of the week as the least occurrence of accident with a total of 1242 accidents.

b. **Significant Hours Analysis**

```
# creating a bar plot
plt.bar(pedestrians.time, pedestrians.Number_of_Accidents)

#Adding a title
plt.title("Number of Accidents based on Hour of the day for pedestrians")

# Adding labels to the axes

plt.xlabel("Hours")
plt.ylabel("Number of Accidents")

# Showing the plot
plt.show()
```



The chart above shows that the most accidents involving pedestrians occurred at 7:22 AM with a total of 2543. With 6:48AM having the least occurrence of accident with the total of 1242 accident.

4. To explore the impact of some feature which contribute to accident severity I had to select some column from accident table, vehicle table and casualty table. Some of the feature selected are light conditions, number of causalities, weather conditions and some other features.

```sql
SELECT accident.accident_year, accident.day_of_week,accident.date,
accident.accident_severity,accident.light_conditions,accident.number_of_casualties,
accident.weather_conditions,vehicle.vehicle_type,
casualty.casualty_severity,casualty.casualty_type
FROM accident
JOIN vehicle ON accident.accident_index = vehicle.accident_index
JOIN casualty ON accident.accident_index = casualty.accident_index
WHERE accident.accident_year = 2020;
```

	accident_year	day_of_week	date	accident_severity	light_conditions	number_of_casualties	weather_conditions	vehicle_type	casualty_severity	casualty_type
1	2020	3	04/02/2020	3	1	1	9	9	3	0
2	2020	2	27/04/2020	3	1	2	1	9	3	0
3	2020	2	27/04/2020	3	1	2	1	9	3	0
4	2020	4	01/01/2020	3	4	1	1	9	3	0
5	2020	4	01/01/2020	2	4	1	1	8	2	0
6	2020	4	01/01/2020	3	4	2	1	9	3	0
7	2020	4	01/01/2020	3	4	2	1	9	3	0
8	2020	4	01/01/2020	3	4	1	1	9	3	9

Imported the necessary libraries using apriori algorithm from mxtend library and the code when run in Google Collab.

```
import pandas as pd
import numpy as np
from mlxtend.frequent_patterns import apriori, association_rules
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Ple
  and should_run_async(code)
Mounted at /content/drive
```

```
data = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/question 4.csv")
data.head()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Ple
  and should_run_async(code)
```

	accident_year	day_of_week	date	accident_severity	light_conditions	number_of_casualties	weather_conditions	vehicle_type	casualty_severity	casualty_type
0	2020	3	04/02/2020	3	1	1	9	9	3	0
1	2020	2	27/04/2020	3	1	2	1	9	3	0
2	2020	2	27/04/2020	3	1	2	1	9	3	0
3	2020	4	01/01/2020	3	4	1	1	9	3	0
4	2020	4	01/01/2020	2	4	1	1	8	2	0

```
data.isnull().sum()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automat
  and should_run_async(code)
accident_year           0
day_of_week             0
date                    0
accident_severity       0
light_conditions        0
number_of_casualties    0
weather_conditions      0
vehicle_type            0
casualty_severity       0
casualty_type           0
dtype: int64
```

```
data = data.drop(['date','accident_year'],axis=1)
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automat
  and should_run_async(code)
```

```
data['accident_severity'].value_counts()
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automat
  and should_run_async(code)
3    171376
2     44828
1      4231
Name: accident_severity, dtype: int64
```

```
#getting the first 1000 rows
data = data.iloc[:1000]
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automat
```

```python
acci1 = (data[data['accident_severity']==1]
         .groupby(['weather_conditions', 'vehicle_type'])['light_conditions']
         .sum().unstack().reset_index().fillna(0)
         .set_index('weather_conditions'))

acci2 = (data[data['accident_severity']==2]
         .groupby(['weather_conditions', 'vehicle_type'])['light_conditions']
         .sum().unstack().reset_index().fillna(0)
         .set_index('weather_conditions'))

acci3 = (data[data['accident_severity']==3]
         .groupby(['weather_conditions', 'vehicle_type'])['light_conditions']
         .sum().unstack().reset_index().fillna(0)
         .set_index('weather_conditions'))
```

```python
def hot_encode(x):
  if(x<= 0):
    return 0
  if(x>= 1):
    return 1

# Encoding the datasets
acci1_encoded = acci1.applymap(hot_encode)
acci1 = acci1_encoded

acci2_encoded = acci2.applymap(hot_encode)
acci2 = acci2_encoded

acci3_encoded = acci3.applymap(hot_encode)
acci3 = acci3_encoded
```

```python
acci2
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: Deprecatio
  and should_run_async(code)
```

vehicle_type	1	2	3	5	8	9	11	19	21
weather_conditions									
1	1	1	1	1	1	1	1	1	1
2	0	0	1	0	0	1	1	0	0
8	0	0	0	0	0	1	0	0	0
9	1	0	0	0	0	0	0	0	0

I used the apriori() method to build the association rule model. Two parameters are required by the function: the DataFrame to be studied and the minimal support criterion. The minimum support level is set to 0.05 in this case, suggesting that an itemset must appear in at least 5% of transactions to be deemed frequent.

```
# Building the model
frq_items = apriori(acci1, min_support = 0.05, use_colnames = True)

# Collecting the inferred rules in a dataframe
rules = association_rules(frq_items, metric = "lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending = [False, False])
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support
  warnings.warn(
```

```
print(rules.head())
```

```
  antecedents consequents  antecedent support  consequent support   support  \
0         (1)        (21)            0.333333            0.666667  0.333333
2         (3)        (21)            0.333333            0.666667  0.333333
1        (21)         (1)            0.666667            0.333333  0.333333
3        (21)         (3)            0.666667            0.333333  0.333333

   confidence  lift  leverage  conviction  zhangs_metric
0         1.0   1.5  0.111111         inf            0.5
2         1.0   1.5  0.111111         inf            0.5
1         0.5   1.5  0.111111    1.333333            1.0
3         0.5   1.5  0.111111    1.333333            1.0
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `
  and should_run_async(code)
```

When examining the resulting table, the emphasis is on the rules' support and confidence. The rule with antecedent (1) and consequent (21) is particularly noteworthy as it has a support of 0.333, meaning that 33.3% of class one accident severity cases occur with it. This implies that changing data values of weather played a significant effect in accidents 33.3% of the time.

```
# Building the model
frq_items = apriori(acci2, min_support = 0.05, use_colnames = True)

# Collecting the inferred rules in a dataframe
rules = association_rules(frq_items, metric = "lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending = [False, False])
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperformance and their support might be dis
  warnings.warn(
```

```
print(rules.head())
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipykernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please pass the result to `transformed_
  and should_run_async(code)
   antecedents consequents  antecedent support  consequent support  support  \
16         (2)         (5)                0.25                0.25     0.25
17         (5)         (2)                0.25                0.25     0.25
18         (8)         (2)                0.25                0.25     0.25
19         (2)         (8)                0.25                0.25     0.25
24         (2)        (19)                0.25                0.25     0.25

    confidence  lift  leverage  conviction  zhangs_metric
16         1.0   4.0    0.1875         inf            1.0
17         1.0   4.0    0.1875         inf            1.0
18         1.0   4.0    0.1875         inf            1.0
19         1.0   4.0    0.1875         inf            1.0
24         1.0   4.0    0.1875         inf            1.0
```

The continuous support for the antecedents (2) and consequents (5) association rule is 0.25. This means that the rule occurs at a 25% frequency. Furthermore, the presence of meteorological circumstances of data values 1, 2, and 3, as well as vehicle type data values 10 and 20, increases the accident severity of data value 2.

```
# Building the model
frq_items = apriori(acci3, min_support = 0.05, use_colnames = True)

# Collecting the inferred rules in a dataframe
rules = association_rules(frq_items, metric = "lift", min_threshold = 1)
rules = rules.sort_values(['confidence', 'lift'], ascending = [False, False])
```

```
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please
  and should_run_async(code)
/usr/local/lib/python3.10/dist-packages/mlxtend/frequent_patterns/fpcommon.py:110: DeprecationWarning: DataFrames with non-bool types result in worse computationalperforman
  warnings.warn(
```

```
print(rules.head())
```

```
   antecedents consequents  antecedent support  consequent support   support  \
82        (17)         (4)            0.142857            0.142857  0.142857
83         (4)        (17)            0.142857            0.142857  0.142857
90        (97)         (4)            0.142857            0.142857  0.142857
91         (4)        (97)            0.142857            0.142857  0.142857
92        (98)         (4)            0.142857            0.142857  0.142857

    confidence  lift  leverage  conviction  zhangs_metric
82         1.0   7.0  0.122449         inf            1.0
83         1.0   7.0  0.122449         inf            1.0
90         1.0   7.0  0.122449         inf            1.0
91         1.0   7.0  0.122449         inf            1.0
92         1.0   7.0  0.122449         inf            1.0
/usr/local/lib/python3.10/dist-packages/ipykernel/ipkernel.py:283: DeprecationWarning: `should_run_async` will not call `transform_cell` automatically in the future. Please
  and should_run_async(code)
```

The continuous support for the association rule with antecedent (17) and consequent (4) is 0.143. This indicates the rule occurs 14.3% of the time. Notably, the occurrence of data values (class) 1 and 2, as well as vehicle type data values (classes) 17 and 20, has an impact on the accident severity of data value 3.

5. I queried the database to bring out the appropriate data needed for the clustering modelling. By joining the Accidents and LSOA table together and filtering accident severity, location, day of the week, time, weather conditions, and lighting conditions in the region. The region I work with are: Kingston upon Hull, Birmingham, Newcastle-under-Lyme and East Riding of Yorkshire.

```sql
SELECT lsoa.lsoa01nm AS 'Location',
       accident.accident_severity AS 'accident_severity',
       accident.day_of_week AS 'day_of_week',
       accident.time,
       accident.weather_conditions AS 'weather_conditions',
       accident.light_conditions AS 'light_conditions',
       accident.accident_year AS 'Year'
FROM accident
JOIN lsoa
ON accident.lsoa_of_accident_location = lsoa.lsoa01cd
WHERE accident.accident_year = 2020
 AND (lsoa.lsoa01nm LIKE 'Kingston upon Hull%'
       OR lsoa.lsoa01nm LIKE 'Haringey%'
       OR lsoa.lsoa01nm LIKE 'Birmingham%'
       OR lsoa.lsoa01nm LIKE 'Newcastle-under-Lyme%'
       OR lsoa.lsoa01nm LIKE 'East Riding of Yorkshire%')
ORDER BY lsoa.lsoa01nm;
```

	Location	accident_severity	day_of_week	time	weather_conditions	light_conditions	Year
3440	Newcastle-und...	1	4	19:25	1	6	2020
3441	Newcastle-und...	3	7	17:07	1	1	2020
3442	Newcastle-und...	3	5	12:03	1	1	2020
3443	Newcastle-und...	3	2	15:00	1	1	2020
3444	Newcastle-und...	3	4	09:19	1	1	2020
3445	Newcastle-und...	3	1	13:14	1	1	2020

```
Execution finished without errors.
Result: 3445 rows returned in 29058ms
```

In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
e).
```

In [3]:
```python
data = pd.read_csv('/content/drive/MyDrive/Quincy/question 5.csv')
data.head()
```

Out[3]:

	Location	accident_severity	day_of_week	time	weather_conditions	light_conditions	Year
0	Birmingham 002B	3	6	08:55	1	1	2020
1	Birmingham 002B	3	4	18:58	1	4	2020
2	Birmingham 002D	3	1	17:34	8	4	2020
3	Birmingham 002D	2	7	16:18	1	1	2020
4	Birmingham 002D	3	2	14:25	1	1	2020

Data cleaning and Pre-processing

In [4]:
```
data['Location'].value_counts()
```

Out[4]:
```
Kingston upon Hull 016D         27
Haringey 011D                   25
Kingston upon Hull 020B         22
Haringey 023C                   21
Haringey 016B                   20
                                ..
East Riding of Yorkshire 004B    1
East Riding of Yorkshire 021C    1
Birmingham 075C                  1
East Riding of Yorkshire 004A    1
Birmingham 074D                  1
Name: Location, Length: 963, dtype: int64
```

In [5]:
```
# Remove the last four characters or numbers from each column
data['Location'] = data['Location']. str[:-4]
data.head()
```

Out[5]:

	Location	accident_severity	day_of_week	time	weather_conditions	light_conditions	Year
0	Birmingham	3	6	08:55	1	1	2020
1	Birmingham	3	4	18:58	1	4	2020
2	Birmingham	3	1	17:34	8	4	2020
3	Birmingham	2	7	16:18	1	1	2020
4	Birmingham	3	2	14:25	1	1	2020

In [6]:
```
data['Location'].value_counts()
```
Out[6]:
```
Birmingham                    1558
Haringey                       734
Kingston upon Hull             569
East Riding of Yorkshire       488
Newcastle-under-Lyme            96
Name: Location, dtype: int64
```
In [7]:
```
data.dtypes
```
Out[7]:
```
Location              object
accident_severity      int64
day_of_week            int64
time                  object
weather_conditions     int64
light_conditions       int64
Year                   int64
dtype: object
```
In [8]:
```
# Check for null Values
data.isnull().sum()
```
Out[8]:
```
Location              0
accident_severity     0
day_of_week           0
time                  0
weather_conditions    0
light_conditions      0
Year                  0
dtype: int64
```
In [9]:
```
data1 = data.copy()
```
In [10]:
```
# Convert the time column to int
data1['time'] = pd.to_datetime(data1['time']).dt.hour
data1['time']
```
Out[10]:
```
0        8
1       18
2       17
3       16
4       14
        ..
3440    17
3441    12
3442    15
3443     9
3444    13
Name: time, Length: 3445, dtype: int64
```

In [11]:
```
from sklearn.cluster import KMeans
from sklearn.preprocessing import normalize , LabelEncoder

# Create a label encoder
le = LabelEncoder()

# Encode the categorical data
data1['Location Encoded'] = le.fit_transform(data1['Location'])
data1
```

Out[11]:

	Location	accident_severity	day_of_week	time	weather_conditions	light_conditions	Year	Location Encoded
0	Birmingham	3	6	8	1	1	2020	0
1	Birmingham	3	4	18	1	4	2020	0
2	Birmingham	3	1	17	8	4	2020	0
3	Birmingham	2	7	16	1	1	2020	0
4	Birmingham	3	2	14	1	1	2020	0
...	...	...	...	...	...	...	...	...
3440	Newcastle-under-Lyme	3	7	17	1	1	2020	4
3441	Newcastle-under-Lyme	3	5	12	1	1	2020	4
3442	Newcastle-under-Lyme	3	2	15	1	1	2020	4
3443	Newcastle-under-Lyme	3	4	9	1	1	2020	4
3444	Newcastle-under-Lyme	3	1	13	1	1	2020	4

3445 rows × 8 columns

In [12]:
```
data2 = data1
data1 = data1.drop('Location', axis=1)
```

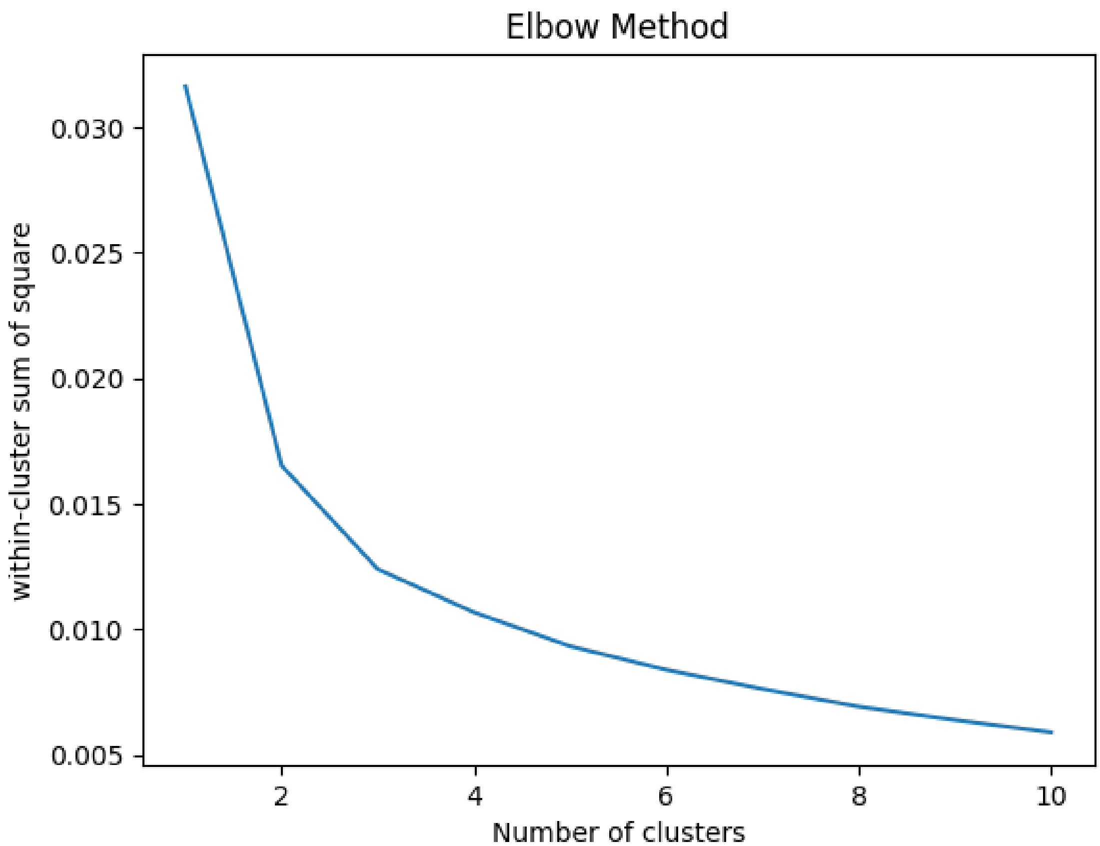
In [13]:
```
# Normalize the data
data2 = data2.drop('Location', axis=1)
data2 = normalize(data2)
```

In [14]:
```
data2
```

Out[14]:
```
array([[1.48512831e-03, 2.97025663e-03, 3.96034217e-03, ...,
        4.95042772e-04, 9.99986399e-01, 0.00000000e+00],
       [1.48508191e-03, 1.98010922e-03, 8.91049148e-03, ...,
        1.98010922e-03, 9.99955154e-01, 0.00000000e+00],
       [1.48507955e-03, 4.95026516e-04, 8.41545077e-03, ...,
        1.98010606e-03, 9.99953562e-01, 0.00000000e+00],
       ...,
       [1.48510193e-03, 9.90067952e-04, 7.42550964e-03, ...,
        4.95033976e-04, 9.99968632e-01, 1.98013590e-03],
       [1.48512595e-03, 1.98016793e-03, 4.45537785e-03, ...,
        4.95041983e-04, 9.99984806e-01, 1.98016793e-03],
       [1.48511266e-03, 4.95037555e-04, 6.43548821e-03, ...,
        4.95037555e-04, 9.99975861e-01, 1.98015022e-03]])
```

```python
In [15]: def elbow_method(data2, max_clusters):
           """
           Calculates the within-cluster sum of squares for different values of k.
           Plots the within-cluster sum of squares against k and returns the optimal value of k.

           Args:
               data: The data to cluster.
               max_clusters: The Maximum Number of clusters to consider.

           Returns:
               The Optimal value of k.
           """
           within_cluster_sum_of_squares = []
           for k in range(1, max_clusters + 1):
             kmeans = KMeans(n_clusters=k, random_state=0).fit(data2)
             within_cluster_sum_of_squares.append(kmeans.inertia_)

           plt.plot(range(1, max_clusters + 1), within_cluster_sum_of_squares)
           plt.title("Elbow Method")
           plt.xlabel('Number of clusters')
           plt.ylabel('within-cluster sum of square')
           plt.show()

           return np.argmin(within_cluster_sum_of_squares) + 1

         if __name__ == "__main__":
           data3 = np.random.randint (0, 100, (100, 2))
           optimal_k = elbow_method(data2, 10)
           print("The optimal number of clusters is:", optimal_k)
```

The elbow point is the optimal number of clusters for the data, and it is at this point that the reduction in the within-cluster sum of squares begins to decelerate. The function illustrates the within-cluster sum of squares for various values of k, revealing that the elbow point lies at k = 3.

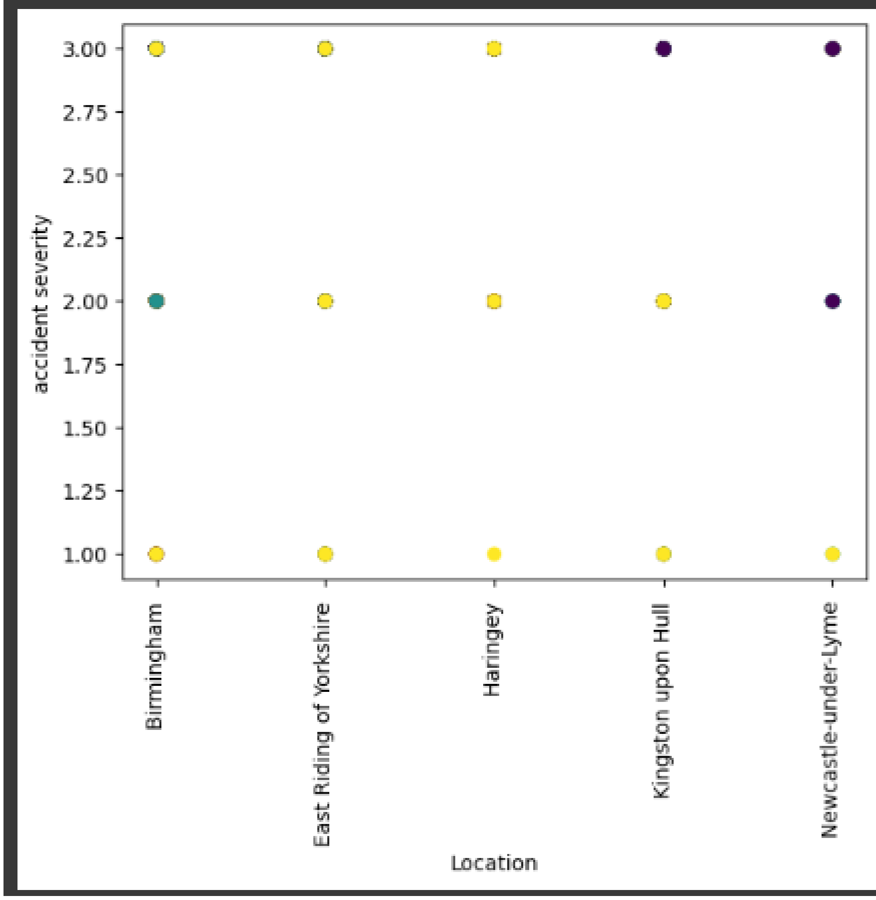


I built a KMeans model with three clusters. I trained the model to the data and used it to predict the cluster labels for each data point.

```
# Create a Kmeans model with 3 cluster
model = KMeans(n_clusters = 3)

# Fit the model to the data
model.fit(data2)

/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning:
  warnings.warn(
▾       KMeans
KMeans(n_clusters=3)

# Predict the cluster labels for each data point
labels = model.predict(data2)

data1['clusters'] = model.labels_
```

Visualising the clustering of the location and accident severity using scatter plot

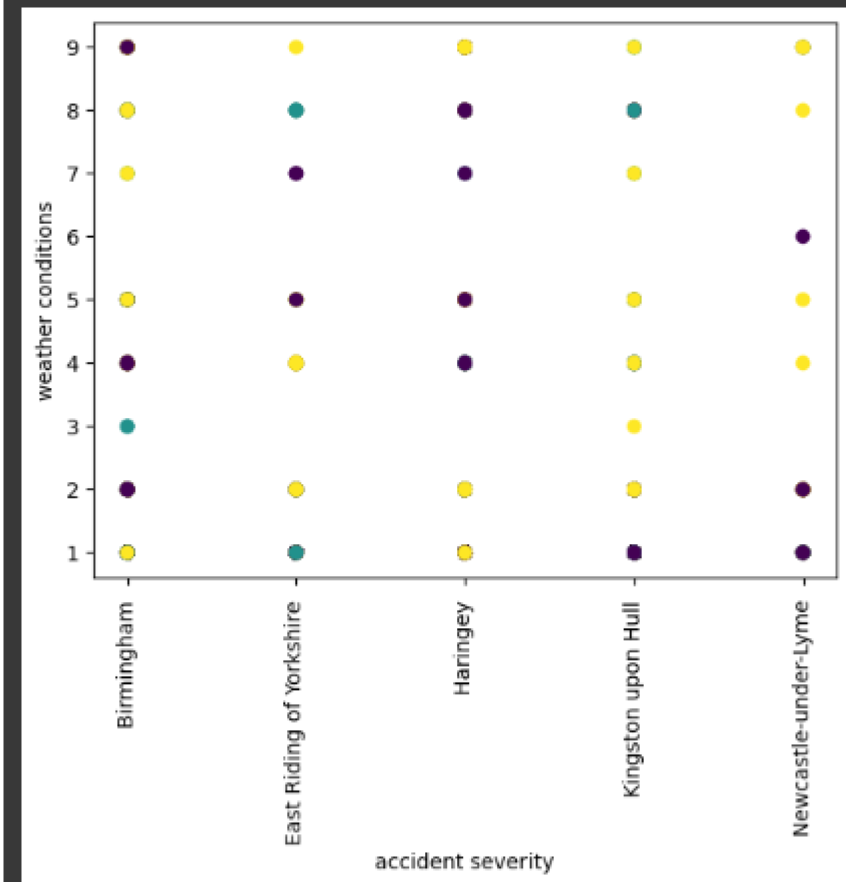
```
# Visualize the clusters using scatter plot
plt.scatter(data['Location'], data1['accident_severity'], c= data1['clusters'])
plt.xlabel('Location')
plt.ylabel('accident severity')
plt.xticks(rotation=90)
plt.show()
```



The scatter plot shows that there is a positive correlation between the severity of accidents and location. This means that cities with more severe accidents tend to be located in more urban areas.

Visualising accident severity and weather conditions to show the correlation below

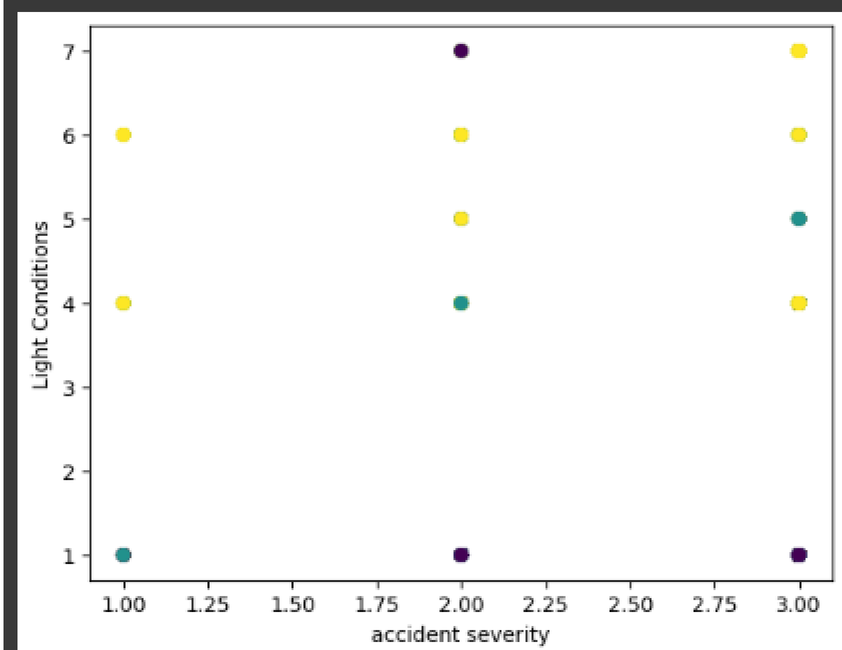
```
# Visualize the clusters using scatter plot
plt.scatter(data['Location'], data1['weather_conditions'], c= data1['clusters'])
plt.xlabel('accident severity')
plt.ylabel('weather conditions')
plt.xticks(rotation=90)
plt.show()
```



The majority of the points in the scatter plot are centred on the graph. This implies that the majority of incidents occur in normal weather conditions. The graph demonstrates a favourable relationship between meteorological conditions and accident severity. This means that accidents are more likely to be severe in adverse weather. There are a few outliers to this pattern, such as weather conditions of 3. These weather conditions have a low accident rate.

Visualising the relationship between accident severity and light circumstances, as well as how light conditions affect accident severity. This scatter plot illustrates that in low-light settings, accidents are more likely to be severe. This tendency has a few deviations, such as light conditions of 2 and 3. Accidents are uncommon under these light circumstances.

```python
# Visualize the clusters using scatter plot
plt.scatter(data['accident_severity'], data1['light_conditions'], c= data1['clusters'])
plt.xlabel('accident severity')
plt.ylabel('Light Conditions')
plt.show()
```



6. I decide to put some important column in the dataset that I think are relevant for the cause of the analysis using SQL syntax.

```sql
SELECT accident.time,
       accident.date,
       accident.accident_severity AS 'accident_severity',
       accident.day_of_week AS 'day_of_week',
       accident.weather_conditions AS 'weather_conditions',
       accident.light_conditions AS 'light_conditions',
       casualty.casualty_severity AS 'casualty_severity',
       casualty.casualty_class AS 'casualty_class',
       casualty.age_of_casualty AS 'age_of_casualty',
       vehicle.vehicle_type AS 'vehicle_type',
       casualty.age_band_of_casualty AS 'age_band_of_casualty',
       accident.road_surface_conditions AS 'road_surface_conditions',
       accident.accident_year AS 'Year'
FROM accident
JOIN vehicle ON accident.accident_index = vehicle.accident_index
JOIN casualty ON accident.accident_index = casualty.accident_index
WHERE accident.accident_year = 2020;
```

	date	accident_severity	day_of_week	weather_conditions	light_conditions	casualty_severity	casualty_class	age_of_casualty	vehicle_type	age_band_of_casualty	road_surface
1	04/02/2020	3	3	9	1	3	3	31	9	6	
2	27/04/2020	3	2	1	1	3	3	2	9	1	
3	27/04/2020	3	2	1	1	3	3	4	9	1	
4	01/01/2020	3	4	1	4	3	3	23	9	5	
5	01/01/2020	2	4	1	4	2	3	47	8	8	

```
Execution finished without errors.
Result: 220435 rows returned in 154340ms
```

In [2]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```python
data = pd.read_csv('/content/drive/MyDrive/Quincy/question 6.csv')
data.head()
```

Out[3]:

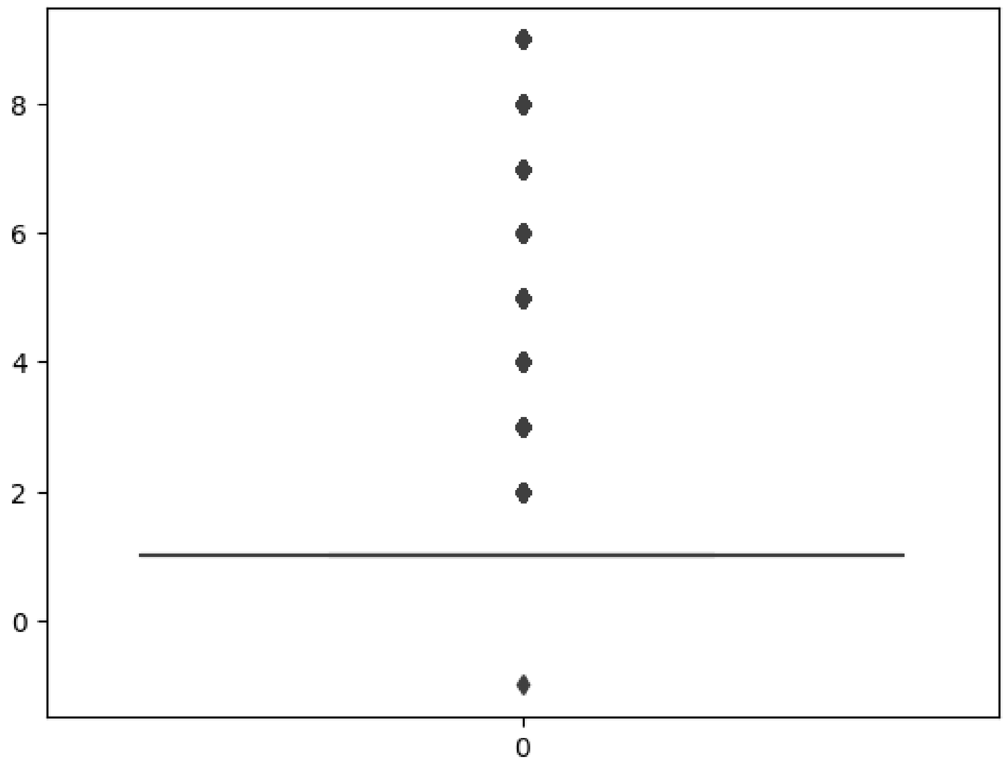
	time	date	accident_severity	day_of_week	weather_conditions	light_conditions	casualty_severity	casualty_class	age_of_casualty	vehicle_type	age_
0	09:00	04/02/2020	3	3	9	1	3	3	31	9	
1	13:55	27/04/2020	3	2	1	1	3	3	2	9	
2	13:55	27/04/2020	3	2	1	1	3	3	4	9	
3	01:25	01/01/2020	3	4	1	4	3	3	23	9	
4	01:50	01/01/2020	2	4	1	4	2	3	47	8	

In [5]:
```python
data.isnull().sum()
```

Out[5]:
```
time                       0
date                       0
accident_severity          0
day_of_week                0
weather_conditions         0
light_conditions           0
casualty_severity          0
casualty_class             0
age_of_casualty            0
vehicle_type               0
age_band_of_casualty       0
road_surface_conditions    0
Year                       0
dtype: int64
```

Using the box plot to check for outliers in the weather conditions

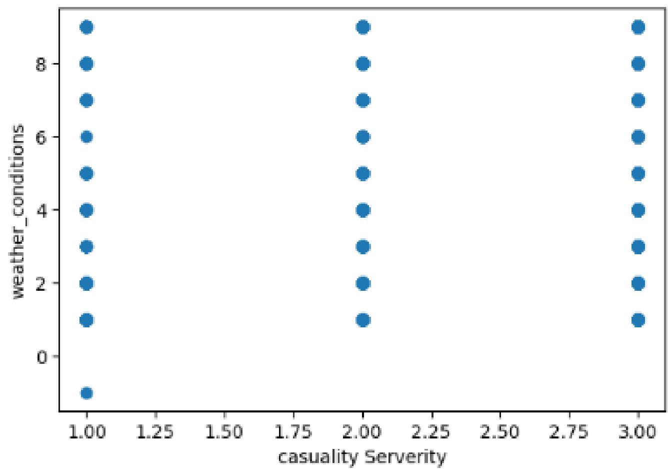
In [6]:
```
# Box plot
sns.boxplot(data['weather_conditions'])
```



The chart shows that there are a lot of outliers in the weather conditions showing serious weather conditions. It shows outliers from class two to class 8 and has a weather class below 0. This might have contributed to the accident rates and casualty severity. So, these outliers need to be kept.

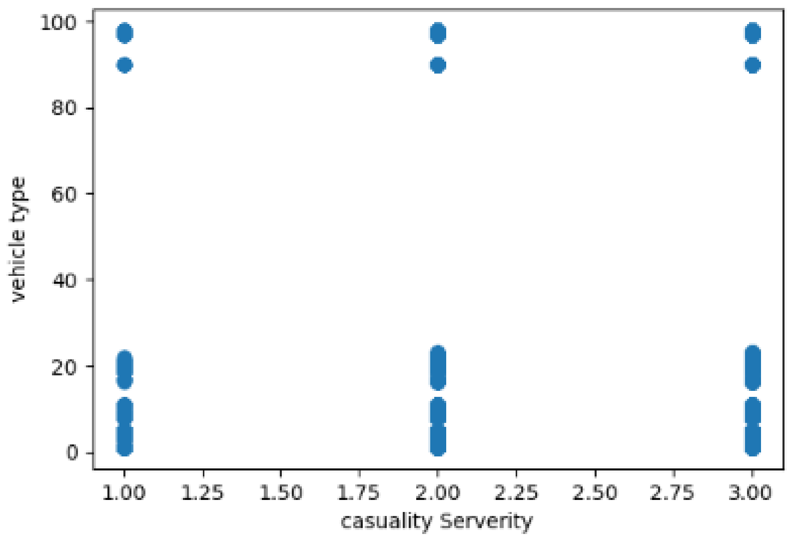
This is because, increase in accident rates can be greatly impacted by severe weather conditions.

```
In [8]: # Scatter plot
fig, ax = plt.subplots(figsize = (6,4))
ax.scatter(data['casualty_severity'],data['weather_conditions'])

# x-axis label
ax.set_xlabel('casuality Serverity')

# y-axis label
ax.set_ylabel('weather_conditions')
plt.show()
```



There is a link between casualty severity and weather conditions. This means that inclement weather increases the likelihood of serious accidents. Rainy weather is the most dangerous, followed by cloudy weather, and finally sunny weather.

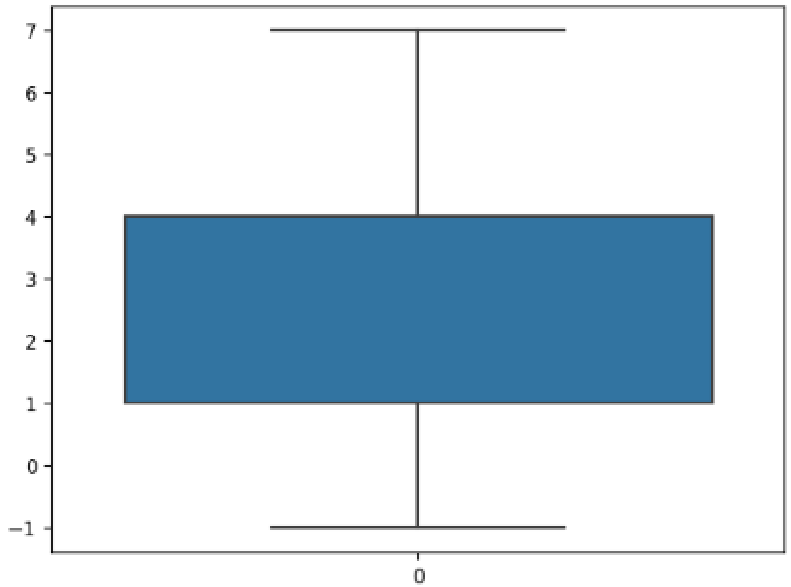
In [10]:
```
# Scatter plot
fig, ax = plt.subplots(figsize = (6,4))
ax.scatter(data['casualty_severity'],data['vehicle_type'])

# x-axis label
ax.set_xlabel('casuality Serverity')

# y-axis label
ax.set_ylabel('vehicle type')
plt.show()
```



There is a link between casualty severity and vehicle type. This indicates that accidents involving larger vehicles, such as trucks and buses, are more likely to be serious than accidents involving smaller vehicles, such as passenger cars. Although there are rare outliers with smaller vehicles causing fatal injuries, this could be due to a variety of other variables such as weather or road surface..

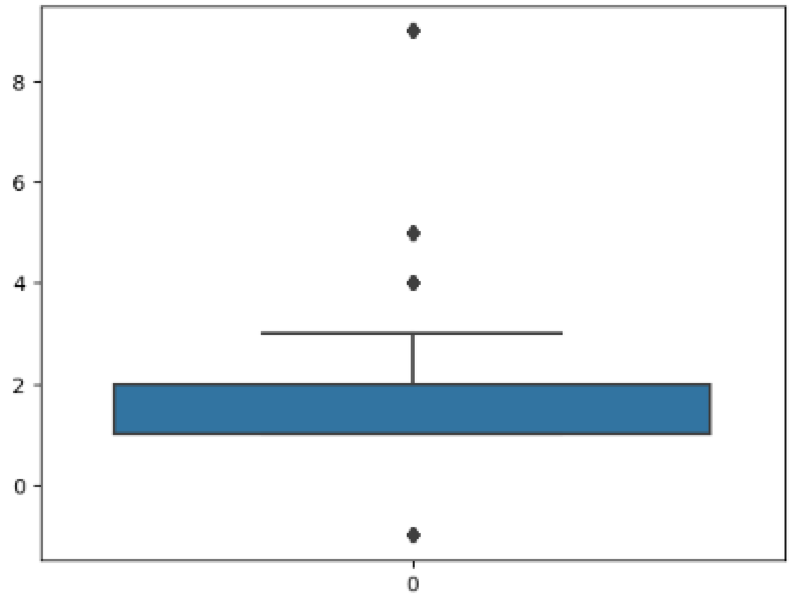
In [11]:
```
# Box Plot
sns.boxplot(data['light_conditions'])
```

Out[11]: <Axes: >



This demonstrates that the distribution of light conditions in the box plot is biassed to the right. This signifies that the majority of the values are concentrated towards the bottom of the distribution, with a long tail stretching to the right. There are no outliers in this column; the right-skewed distribution of the light conditions box plot is most likely due to the fact that there are more elements that can contribute to poor performance in low light than in brilliant light.

```
In [13]: sns.boxplot(data['road_surface_conditions'])

Out[13]: <Axes: >
```



The distribution of road surface conditions is skewed to the right in this box plot. This signifies that the majority of the values are concentrated towards the bottom of the distribution, with a long tail stretching to the right. Outliers range from data value 4 outside the box plot graph to an outlier down the box plot.

Prediction

7. The python libraries to be used for data cleaning, data pre-processing and the machine learning model were imported

In [1]:
```
# import Libaries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
```

In [2]:
```
data = pd.read_csv('/content/drive/MyDrive/Quincy/question 7.csv')
```

In [3]:
```
data.head(5)
```

Out[3]:

	time	date	accident_severity	day_of_week	weather_conditions	light_conditions	casualty_severity	casualty_class	age_of_casualty	vehicle_type	age_
0	09:00	04/02/2020	3	3	9	1	3	3	31	9	
1	13:55	27/04/2020	3	2	1	1	3	3	2	9	
2	13:55	27/04/2020	3	2	1	1	3	3	4	9	
3	01:25	01/01/2020	3	4	1	4	3	3	23	9	
4	01:50	01/01/2020	2	4	1	4	2	3	47	8	

In [4]:
```
data.isnull().sum()
```

Out[4]:
```
time                       0
date                       0
accident_severity          0
day_of_week                0
weather_conditions         0
light_conditions           0
casualty_severity          0
casualty_class             0
age_of_casualty            0
vehicle_type               0
age_band_of_casualty       0
road_surface_conditions    0
Year                       0
dtype: int64
```

In [5]:
```
data.dtypes
```

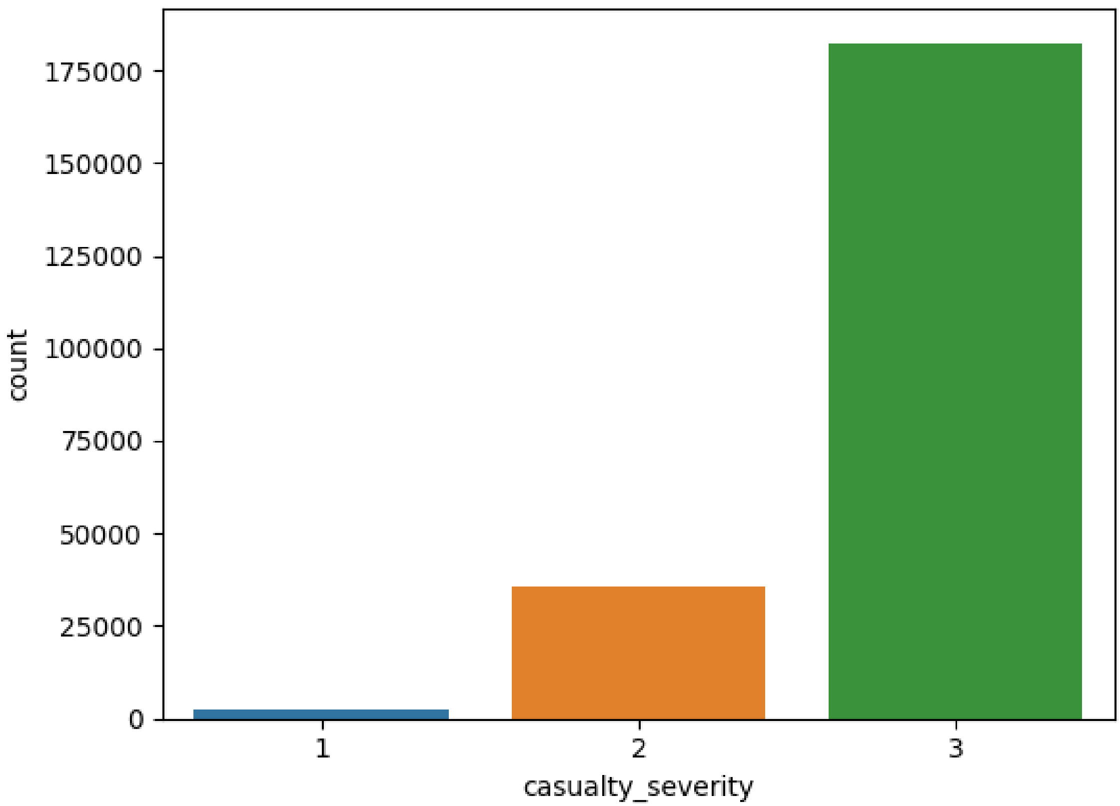
Out[5]:
```
time                       object
date                       object
accident_severity           int64
day_of_week                 int64
weather_conditions          int64
light_conditions            int64
casualty_severity           int64
casualty_class              int64
age_of_casualty             int64
vehicle_type                int64
age_band_of_casualty        int64
road_surface_conditions     int64
Year                        int64
dtype: object
```

In [6]:
```
# convert the time column to int
data['time'] = pd.to_datetime(data['time']).dt.hour
data['date'] = pd.to_datetime(data['time']).dt.hour
```

In [6]:
```
# convert the time column to int
data['time'] = pd.to_datetime(data['time']).dt.hour
data['date'] = pd.to_datetime(data['time']).dt.hour
```


In [7]:
```
# Checking if there is imbalance in the target variable
sns.countplot(x = 'casualty_severity', data = data)
```

I observed the data values in the dataset were not evenly distributed.

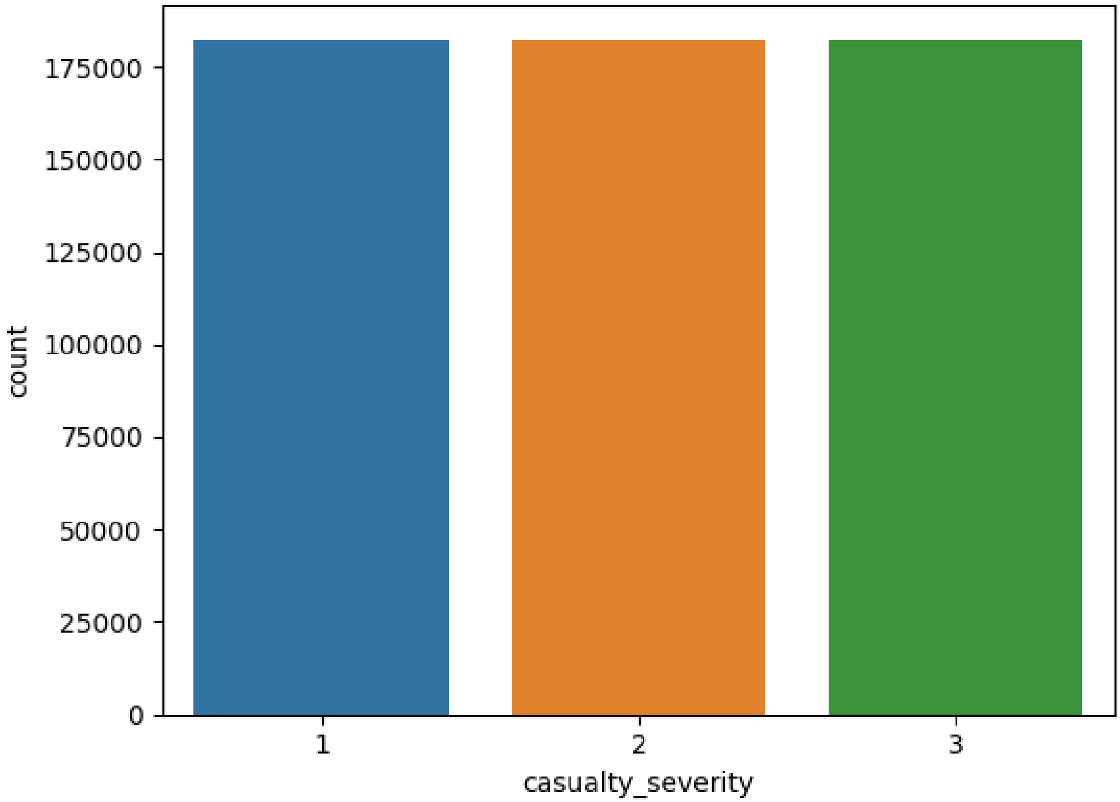


I resampled the dataset using the sampling SMOTE package to make the target variable more uniformly distributed.

```
smote = SMOTE(random_state = 10)
data, data['casualty_severity'] = smote.fit_resample(data, data['casualty_severity'])
```

```
# Check again to see if it has been resampled to an evenly distribution
sns.countplot(x = 'casualty_severity', data = data)
```

Then I checked the distribution of the data values again, which showed an even distribution of the column labels as shown below.



In [10]:
```
y = data['casualty_severity']
data = data.drop(['casualty_severity','Year'], axis = 1)
```

In [11]:
```
# Scale the data
from sklearn.model_selection import train_test_split
scaler = StandardScaler()
data = scaler.fit_transform(data)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size = 0.25)
```

In [12]:
```
# Create a logistic regression model
model = LogisticRegression()

# Define the hyperparameters that you want to tune
param_grid = {'penalty': ['l1','l2'], 'C': [0.01, 0.1, 1, 10, 100]}
```

In [13]:
```
# Use GridSearchCV to find the best hyperparameters for your model
grid_search = GridSearchCV(model, param_grid, cv=5)
grid_search.fit(X_train, y_train)
```

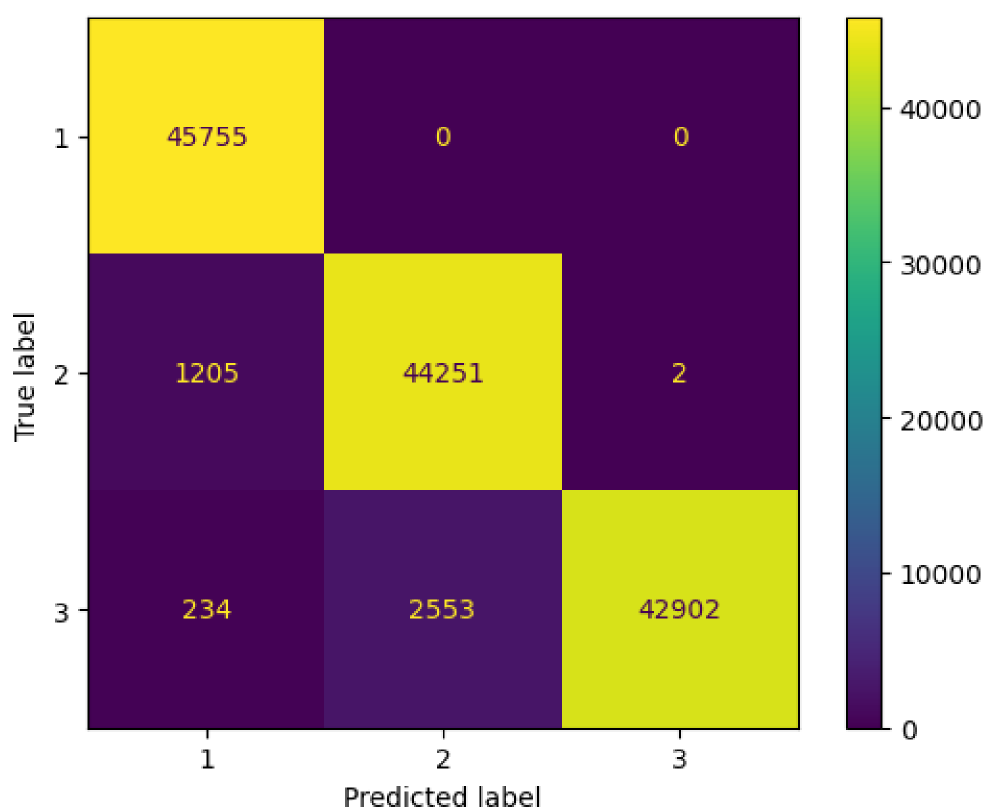
Out[13]:
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1, 10, 100], 'penalty': ['l1', 'l2']})
```

Then the model was used to predict the target variables and also two different metrics were performed on it. The accuracy which brought out 97% shows the model performed well in predicting the target variable. I used the confusion matrix to get deeper on the model performance.

```
In [14]: from sklearn.model_selection import train_test_split, GridSearchCV
         from sklearn.preprocessing import StandardScaler
         from sklearn.linear_model import LogisticRegression


         # Use the best model to make predictions on the test set
         y_pred = grid_search.predict(X_test)
```

```
In [15]: from sklearn.metrics import accuracy_score
         accuracy_score(y_test, y_pred)

Out[15]: 0.9708258462257673
```

```
In [16]: # Confusion matrix
         cm= confusion_matrix(y_test, y_pred, labels=grid_search.classes_)

         CMD= ConfusionMatrixDisplay(confusion_matrix=cm,
                                     display_labels = grid_search.classes_)

         CMD.plot()
```



I utilised the confusion matrix, then constructed a Confusion Matrix Display object, and lastly plotted the confusion matrix. This demonstrates that the model correctly classified one of the target variables. However, the model misclassified certain data points in the second and third classes.

RECOMMENDATION

Throughout the majority of my research, I discovered that there is a direct association between weather and lighting conditions and accidents, so some safety precautions can be done. This include:

In stormy weather:

- Drive more cautiously and more slowly.
- Increasing the distance, you follow.
- Even during the day, turn on your headlights.
- Be ready for slick driving conditions.
- Driving should be avoided in severe weather.

REFERENCE

1. Agho, E.Q (2023), Big Data and Data Mining Project Report, Big Data and Data Mining, 771763_B22_T3A, University of Hull, UK.
2. Divvela, S.R (2020) Apriori Algorithms in Data Mining| Algorithms with Example. Available Online: https://www.youtube.com/watch?v=FLm3pxBtTaU
3. Emmys, C. (2023) *Comprehensive Guide to SQL Fundamentals and Practical Applications*: GitforGits Asian Publishing House, India.
4. James, R.G & Paul, N.W (1999) *SQL: The Complete Reference*: Osborne/McGraw-Hill, California, United States of America.
5. John, A (2020) SQLite Databases with Python: FreeCode.com. Available Online: https://www.youtube.com/watch?v=byHcYRpMgI4
6. Paul,D (2014) *MySQL Cookbook, 3rd Edition*: O'Reilly Media, USA.
7. Reported Road Casualties in Great Britian: Notes, Definitions, Symbols and Conventions. Available Online: https://www.gov.uk/government/publications/road-accidents-and-safety-statistics-notes-and-definitions/reported-road-casualties-in-great-britain-notes-definitions-symbols-and-conventions. [Accessed 19/12/2023]
8. Statistics Record of Road Accidents. Available Online: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/995422/stats19.pdf
9. Steve,S. (2002) MySQL Bible, Wiley Publishing Inc, New York, USA.
10. Trouble-Free Channel (2021) Mining Methods- Apriori Algorithms with Example. Available Online: https://www.youtube.com/watch?v=CcaRfwlHyNw