How Do Road Accidents Cluster Spatially in London Borough and What Traffic Behaviour Influence Accident Severity most?

Preparation

- Github link
- Number of words: 1507
- Runtime: 76 seconds (Memory 32 GB, CPU Intel Ultra 9 @2.50GHz)
- Coding environment: SDS Docker
- License: this notebook is made available under the Creative Commons Attribution license.
- Additional library:
    - **watermark**: A Jupyter Notebook extension for printing timestamps, version numbers, and hardware information.
    - **contextily**: A library for adding basemaps to geospatial visualizations in Python.
    - **pysal**: A library for spatial data analysis and spatial econometrics.
    - **esda**: A library for exploratory spatial data analysis.

Table of contents

1. Introduction
2. Research questions
3. Data
4. Methodology
5. Analysis and discussion
6. Conclusion
7. References

1.Introduction

[go back to the top]

Understanding why some road accidents are more severe than others, and where these accidents happen most, is crucial for making cities like London safer.

A lot of previous research has focused on what factors increase accident severity. For example, Castro and Kim (2016) showed that machine learning models like decision trees and neural networks are good at picking out the key risk factors. Zhu et al. (2024) used logistic regression and random forests to look at what influences injury levels, while Gray et al. (2008) found certain traits make young male drivers more likely to have serious crashes. Marchant and colleagues (2020) studied how factors affect pedestrian injuries at different types of crossings.

However, most of these studies have not looked at the bigger picture—specifically, where in the city severe accidents are more likely to cluster.

By combining spatial analysis with classification models, I want to find out which London boroughs see the most serious accidents and what driving behaviors are most risky.

2.Research questions

[go back to the top]

My research investigates two interrelated questions:

- **RQ1: How do accidents of different severity levels cluster across London boroughs?**
- **RQ2: Which traffic behaviors most significantly influence the severity of road accidents?**

Using the UK Department for Transport's Road Safety Data and London GIS boundary , I will analyze relationships between accident severity (slight, serious, fatal) and various traffic behaviors including age, sex, driver attributes, pedestrain movement, and vehicle type...

3.Data

[go back to the top]

I use five years of London road safety data **from 2019 to 2023**. This time frame is chosen to ensure the analysis reflects recent trends and captures enough accident cases for robust statistical and spatial analysis.

Due to the limitations of the geographic information provided in the road safety data, I use the **London borough** as the spatial unit for clustering analysis. This administrative level is the most detailed and consistent location data available in

the dataset, allowing for meaningful spatial patterns to be identified across the city.

First, operations are performed in Excel: the scientific notation in the collision data is converted to text format, and then data for the London area (E09) is filtered and exported as a new CSV file.

Next, the filtered CSV is right-joined with the casualty and vehicle tables, and the resulting dataset is exported as a new file.

> Note: The exported files already upload to Github, so doesn't need to execute.

In [6]:
```
# import pandas as pd
# import os

# # Read data files
# collision_df = pd.read_csv('data/raw/dft-road-casualty-statistics-collision-la
# casualty_df = pd.read_csv('data/raw/dft-road-casualty-statistics-casualty-last
# vehicle_df = pd.read_csv('data/raw/dft-road-casualty-statistics-vehicle-last-5

# # Select required columns
# collision_columns = [
#     'accident_index',
#     'longitude',
#     'latitude',
#     'accident_severity',
#     'number_of_vehicles',
#     'number_of_casualties',
#     'local_authority_ons_district'
# ]

# casualty_columns = [
#     'accident_index',
#     'casualty_reference',
#     'casualty_class',
#     'sex_of_casualty',
#     'age_of_casualty',
#     'age_band_of_casualty',
#     'casualty_severity'
# ]

# vehicle_columns = [
#     'accident_index',
#     'vehicle_reference',
#     'vehicle_type',
#     'towing_and_articulation',
#     'vehicle_manoeuvre',
#     'vehicle_left_hand_drive',
#     'journey_purpose_of_driver',
#     'sex_of_driver',
#     'age_of_driver',
#     'age_band_of_driver',
#     'engine_capacity_cc',
#     'propulsion_code',
#     'age_of_vehicle'
# ]
```

```
# # Filter data
# collision_df = collision_df[collision_columns]
# casualty_df = casualty_df[casualty_columns]
# vehicle_df = vehicle_df[vehicle_columns]

# # Print original data statistics
# print(f"Original Data Statistics:")
# print(f"Collision records: {len(collision_df)}")
# print(f"Casualty records: {len(casualty_df)}")
# print(f"Vehicle records: {len(vehicle_df)}")

# # Filter London area data
# london_collision_df = collision_df[collision_df['local_authority_ons_district'
# print(f"\nLondon area collision records: {len(london_collision_df)}")

# # Ensure output directory exists
# os.makedirs('data', exist_ok=True)

# # Delete existing files if they exist
# output_files = [
#     'data/dft-london-collision-2019-2023.csv',
#     'data/dft-london-collision-casualty.csv',
#     'data/dft-london-collision-vehicle.csv'
# ]

# for file in output_files:
#     if os.path.exists(file):
#         os.remove(file)
#         print(f"Deleted existing file: {file}")

# # Save London area data
# london_collision_df.to_csv('data/dft-london-collision-2019-2023.csv', index=Fa

# # Merge data
# # Merge casualty and collision data (casualty as primary)
# casualty_collision_df = pd.merge(
#     casualty_df,
#     collision_df,
#     on='accident_index',
#     how='right'  # Use right join to keep all collision data
# )

# # Merge vehicle and collision data (vehicle as primary)
# vehicle_collision_df = pd.merge(
#     vehicle_df,
#     collision_df,
#     on='accident_index',
#     how='right'  # Use right join to keep all collision data
# )

# # Print merged data statistics
# print(f"\nMerged Data Statistics:")
# print(f"Casualty-Collision merged records: {len(casualty_collision_df)}")
# print(f"Vehicle-Collision merged records: {len(vehicle_collision_df)}")

# # Save merged data
# casualty_collision_df.to_csv('data/dft-london-collision-casualty.csv', index=F
# vehicle_collision_df.to_csv('data/dft-london-collision-vehicle.csv', index=Fal
```

By analyzing the proportion of missing values in each column, I deleted columns with a high rate of missing data.

In the end, I selected the variables listed in the table below as the focus of my analysis.

File	Variable	Type
collision	accident_reference	categorical
collision	accident_severity	categorical
collision	number_of_vehicles	numeric
collision	number_of_casualties	numeric
collision	local_authority_ons_district	categorical
casualty	casualty_class	categorical
casualty	sex_of_casualty	categorical
casualty	age_of_casualty	numeric
casualty	casualty_severity	categorical
casualty	pedestrian_location	categorical
casualty	pedestrian_movement	categorical
vehicle	vehicle_reference	categorical
vehicle	vehicle_type	categorical
vehicle	towing_and_articulation	categorical
vehicle	vehicle_manoeuvre	categorical
vehicle	vehicle_left_hand_drive	categorical
vehicle	journey_purpose_of_driver	categorical
vehicle	sex_of_driver	categorical
vehicle	age_of_driver	numeric
vehicle	age_band_of_driver	categorical
vehicle	engine_capacity_cc	numeric
vehicle	propulsion_code	categorical
vehicle	age_of_vehicle	numeric

4.Methodology

[go back to the top]

4.1 DBSCAN

DBSCAN is an unsupervised clustering algorithm that groups together points that are closely packed, and identifies points that lie alone in low-density regions as outliers.

It does not require the number of clusters to be specified beforehand and is especially useful for discovering clusters with irregular shapes.

4.2 Logistic Regression

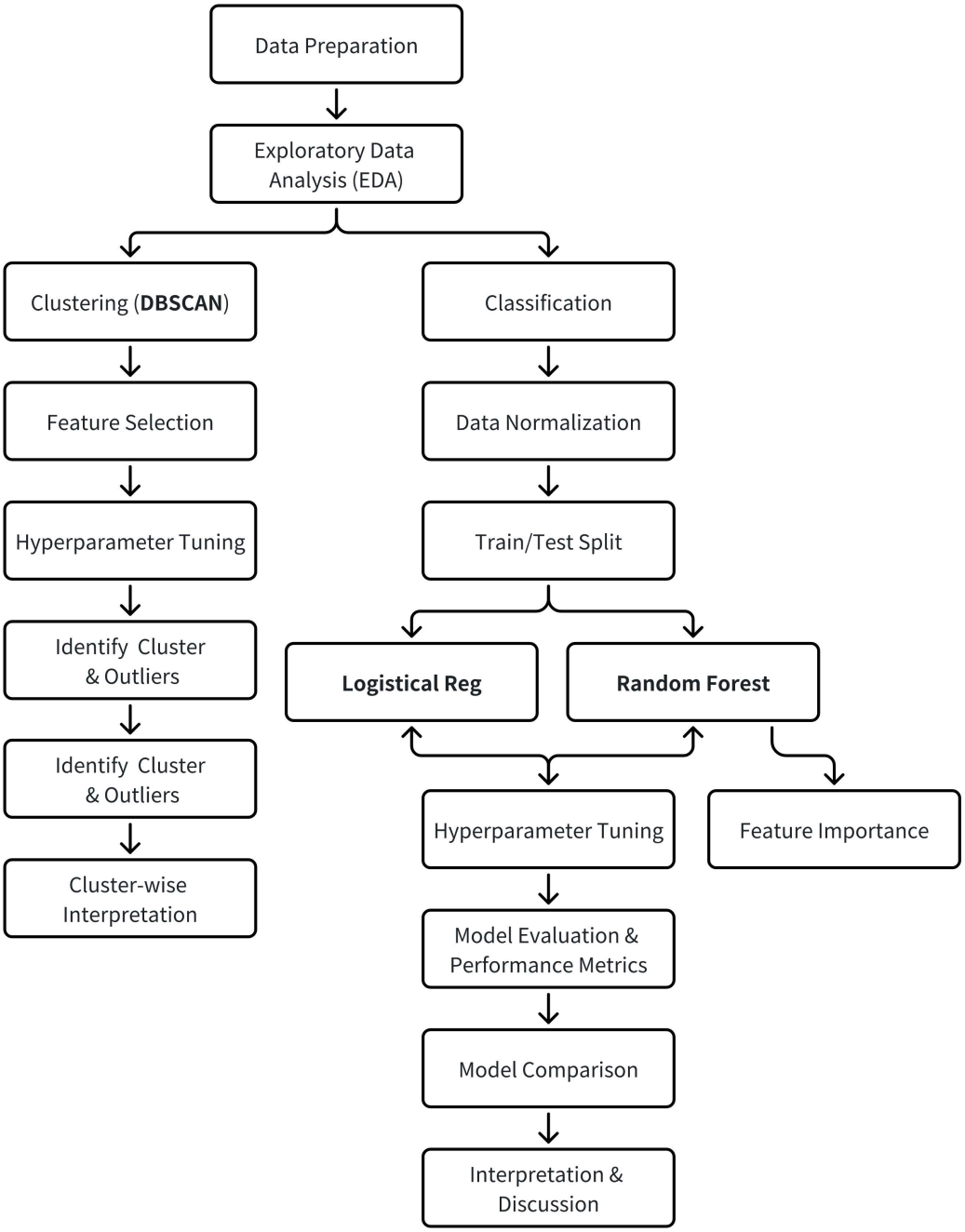
Logistic regression is a supervised learning algorithm used for binary classification tasks. It models the probability that a sample belongs to a certain class using a logistic function. The output is a value between 0 and 1, which is interpreted as the probability of belonging to the positive class.

4.3 Random Forest

Random forest is an ensemble machine learning method used for both classification and regression. It builds multiple decision trees during training and merges their results to improve prediction accuracy and avoid overfitting. Each tree is built from a random subset of the data and features.

4.4 Methodology Flow Chart

The flow chart of the methods used in my research process is as follows:



5.Analysis and discussion

[go back to the top]

These are the packages that will be used throughout the analysis.

In [28]:
```
# !pip install contextily
# !pip install pysal
# !pip install esda
```

In [8]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import chi2_contingency, pointbiserialr
```

```
import geopandas as gpd
from shapely.geometry import Point
import contextily as ctx

# cluster
from sklearn.cluster import DBSCAN
from sklearn.neighbors import NearestNeighbors
from sklearn import metrics
import pysal as ps
from esda.adbscan import ADBSCAN
import matplotlib.colors as colors
import matplotlib.cm as cm
from matplotlib import rcParams

# classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_mat
from sklearn.feature_selection import SelectKBest, f_classif, chi2
```

5.1 Data processing

Clean the data by removing rows with NULL and invalid entries.

In [9]:
```
collision_df = pd.read_csv('https://media.githubusercontent.com/media/Cihshee/CA
casualty_df = pd.read_csv('https://media.githubusercontent.com/media/Cihshee/CAS
vehicle_df = pd.read_csv('https://media.githubusercontent.com/media/Cihshee/CASA

# Print original data information
print("Original Data:")
print("collision_df:", collision_df.shape)
print("casualty_df:", casualty_df.shape)
print("vehicle_df:", vehicle_df.shape)

# Data cleaning function
def clean_dataframe(df):
    # Remove rows with missing values
    df = df.dropna()
    # Remove rows containing -1
    df = df[~(df == -1).any(axis=1)]
    return df

collision_df = clean_dataframe(collision_df)
casualty_df = clean_dataframe(casualty_df)
vehicle_df = clean_dataframe(vehicle_df)

# Clean other invalid data
casualty_df = casualty_df[~casualty_df['sex_of_casualty'].isin([9])]
vehicle_df = vehicle_df[~vehicle_df['vehicle_type'].isin([99])]
vehicle_df = vehicle_df[~vehicle_df['towing_and_articulation'].isin([9])]
vehicle_df = vehicle_df[~vehicle_df['vehicle_manoeuvre'].isin([99])]
vehicle_df = vehicle_df[~vehicle_df['vehicle_left_hand_drive'].isin([9])]
vehicle_df = vehicle_df[~vehicle_df['journey_purpose_of_driver'].isin([6])]
```

```
vehicle_df = vehicle_df[~vehicle_df['journey_purpose_of_driver'].isin([15])]
vehicle_df = vehicle_df[~vehicle_df['sex_of_driver'].isin([3])]

# Print cleaned data information
print("Cleaned Data:")
print("collision_df:", collision_df.shape)
print("casualty_df:", casualty_df.shape)
print("vehicle_df:", vehicle_df.shape)
```

```
<ipython-input-9-03f159cb4991>:3: DtypeWarning: Columns (0) have mixed types. Spe
cify dtype option on import or set low_memory=False.
  vehicle_df = pd.read_csv('https://media.githubusercontent.com/media/Cihshee/CAS
A0006/refs/heads/main/data/dft-london-collision-vehicle.csv')
Original Data:
collision_df: (115805, 7)
casualty_df: (631605, 13)
vehicle_df: (868777, 19)
Cleaned Data:
collision_df: (115804, 7)
casualty_df: (488258, 13)
vehicle_df: (240534, 19)
```

The severity categories are not evenly distributed, which can cause sample bias.

So here, we going to re-encode them into **binary categories.**

In [10]:
```
# Count accident severity and casualty severity distribution
# print(collision_df['accident_severity'].value_counts())
# print(vehicle_df['accident_severity'].value_counts())
# print(casualty_df['casualty_severity'].value_counts())

# Recode severity into binary categories
# 2: Fatal/Severe injury (original 1,2)
# 1: Slight injury (original 3)

collision_df['accident_severity'] = collision_df['accident_severity'].apply(lamb
vehicle_df['accident_severity'] = vehicle_df['accident_severity'].apply(lambda x
casualty_df['casualty_severity'] = casualty_df['casualty_severity'].apply(lambda

# Print the distribution of recoded severity
print(collision_df['accident_severity'].value_counts())
print(vehicle_df['accident_severity'].value_counts())
print(casualty_df['casualty_severity'].value_counts())
```

```
accident_severity
1    98179
2    17625
Name: count, dtype: int64
accident_severity
1    180148
2     60386
Name: count, dtype: int64
casualty_severity
1    392245
2     96013
Name: count, dtype: int64
```

5.2 Spatial cluster analysis

In this section, I will conduct a spatial cluster analysis at the London borough level.

I will use **DBSCAN method** to examine the distribution and clustering of road accidents across different boroughs to better understand spatial patterns and identify areas with higher or lower accident concentrations.

5.2.1 Raw distribution by borough

Check the count of accidents by district and prepare the geographic data for spatial analysis.

In [11]:
```
# Set default font for plots
tfont = {'weight': 'normal', 'size': 14}

# Data Preparation - Check geographic information and administrative district in
print("District distribution:", casualty_df['local_authority_ons_district'].valu
```

```
District distribution: local_authority_ons_district
E08000025    13992
E08000035     6970
E06000054     5849
E08000032     4864
E09000033     4403
Name: count, dtype: int64
```

Create a GeoDataFrame for accident locations and load London borough boundary data.

In [12]:
```
# Create GeoDataFrame based on coordinates
# Ensure coordinates are valid
casualty_gdf = casualty_df.copy()
casualty_gdf = casualty_gdf.dropna(subset=['longitude', 'latitude', 'local_authc
casualty_gdf = casualty_gdf[(casualty_gdf['longitude'] != 0) & (casualty_gdf['la

# Create GeoDataFrame
geometry = [Point(xy) for xy in zip(casualty_gdf['longitude'], casualty_gdf['lat
casualty_gdf = gpd.GeoDataFrame(casualty_gdf, geometry=geometry, crs="EPSG:4326"
print("Created GeoDataFrame:", casualty_gdf.shape)

# Download London borough boundary data
print("\nDownloading London borough boundary data...")
url = 'https://data.london.gov.uk/download/statistical-gis-boundary-files-london
! wget $url

london_boroughs = gpd.read_file(f"zip://statistical-gis-boundaries-london.zip!st
print("London borough boundary data shape:", london_boroughs.shape)
print("London borough boundary data columns:", london_boroughs.columns)
```

```
Created GeoDataFrame: (488258, 14)

Downloading London borough boundary data...
--2025-04-29 11:35:44--  https://data.london.gov.uk/download/statistical-gis-boun
dary-files-london/9ba8c833-6370-4b11-abdc-314aa020d5e0/statistical-gis-boundaries
-london.zip
Resolving data.london.gov.uk (data.london.gov.uk)... 172.67.72.228, 104.26.7.203,
104.26.6.203, ...
Connecting to data.london.gov.uk (data.london.gov.uk)|172.67.72.228|:443... conne
cted.
HTTP request sent, awaiting response... 302 Found
Location: https://airdrive-secure.s3-eu-west-1.amazonaws.com/london/dataset/stati
stical-gis-boundary-files-london/2016-10-03T13%3A52%3A28/statistical-gis-boundari
es-london.zip?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAJJDIMAIVZJDIC
KHA%2F20250429%2Feu-west-1%2Fs3%2Faws4_request&X-Amz-Date=20250429T113545Z&X-Amz-
Expires=300&X-Amz-Signature=e93ac32f2edeedf4ddbfbec90e63d780fec7ed3103292d51a3b27
f3d13dfa7b2&X-Amz-SignedHeaders=host [following]
--2025-04-29 11:35:45--  https://airdrive-secure.s3-eu-west-1.amazonaws.com/londo
n/dataset/statistical-gis-boundary-files-london/2016-10-03T13%3A52%3A28/statistic
al-gis-boundaries-london.zip?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AK
IAJJDIMAIVZJDICKHA%2F20250429%2Feu-west-1%2Fs3%2Faws4_request&X-Amz-Date=20250429
T113545Z&X-Amz-Expires=300&X-Amz-Signature=e93ac32f2edeedf4ddbfbec90e63d780fec7ed
3103292d51a3b27f3d13dfa7b2&X-Amz-SignedHeaders=host
Resolving airdrive-secure.s3-eu-west-1.amazonaws.com (airdrive-secure.s3-eu-west-
1.amazonaws.com)... 52.92.1.10, 3.5.68.229, 3.5.67.159, ...
Connecting to airdrive-secure.s3-eu-west-1.amazonaws.com (airdrive-secure.s3-eu-w
est-1.amazonaws.com)|52.92.1.10|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 28666674 (27M) [application/zip]
Saving to: ‘statistical-gis-boundaries-london.zip’

statistical-gis-bou 100%[===================>]  27.34M  16.3MB/s    in 1.7s

2025-04-29 11:35:47 (16.3 MB/s) - ‘statistical-gis-boundaries-london.zip’ saved
[28666674/28666674]

London borough boundary data shape: (33, 8)
London borough boundary data columns: Index(['NAME', 'GSS_CODE', 'HECTARES', 'NON
LD_AREA', 'ONS_INNER', 'SUB_2009',
       'SUB_2006', 'geometry'],
      dtype='object')
```

Convert coordinate system, assign each accident to a borough, and calculate borough-level accident statistics.

In [13]:

```python
# Convert coordinate reference system
casualty_gdf = casualty_gdf.to_crs(epsg=3857)  # Web Mercator projection
london_boroughs = london_boroughs.to_crs(epsg=3857)

# Create borough-level casualty statistics
print("\nCalculating borough-level accident statistics...")

# Use spatial join to determine which borough each accident belongs to
casualty_with_borough = gpd.sjoin(casualty_gdf, london_boroughs, how='left', pre

# Group by borough and accident severity
borough_severity_stats = casualty_with_borough.groupby(['NAME', 'casualty_severi

borough_severity_stats = borough_severity_stats[[2,1]]  # Reorder columns: 2-Fat
```

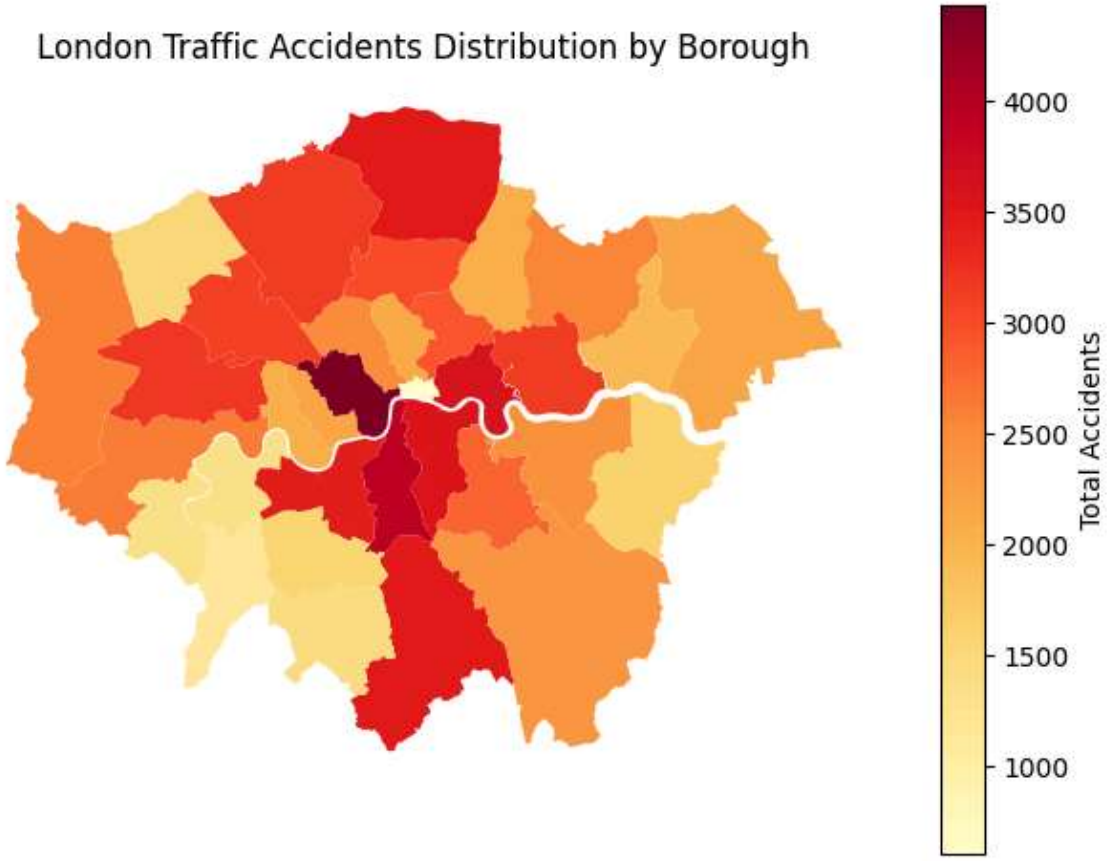
```
borough_severity_stats.columns = ['Fatal and Serious Accidents', 'Slight Acciden

# Calculate total accidents
borough_severity_stats['Total Accidents'] = borough_severity_stats.sum(axis=1)

print(borough_severity_stats.sort_values(by='Total Accidents', ascending=False).
```

```
Calculating borough-level accident statistics...
              Fatal and Serious Accidents  Slight Accidents  Total Accidents
NAME
Westminster                           757              3676             4433
Lambeth                               649              3255             3904
Tower Hamlets                         480              3114             3594
Southwark                             545              3000             3545
Enfield                               369              3133             3502
Croydon                               506              2982             3488
Wandsworth                            555              2889             3444
Ealing                                381              2824             3205
Newham                                419              2738             3157
Barnet                                439              2716             3155
```

Merge accident statistics with borough boundaries. Then draw an **overview heatmap** about the total number of accidents in each borough.

In [14]:
```
# Add statistics to borough boundary file
london_boroughs_with_stats = london_boroughs.merge(borough_severity_stats, left_

# Visualize borough accident distribution heatmap
fig = plt.figure(figsize=(15, 10))

# Total accidents heatmap
ax = london_boroughs_with_stats.plot(column='Total Accidents', legend=True, cmap

ax.set_title('London Traffic Accidents Distribution by Borough')
ax.set_axis_off()

plt.tight_layout()
plt.show()
```

```
<Figure size 1500x1000 with 0 Axes>
```



5.2.2 DBSCAN

Let's run the DBSCAN clustering method. According to the documentation, the model can take a number of parameters: the `eps` distance and `min_samples` group size attributes are fundamental in creating the clusters.

I extract the accident count and average severity as features for clustering.

These features are **normalized** to ensure both have equal importance in the DBSCAN algorithm.

In [15]:
```
# DBSCAN Clustering Analysis Based on Borough Statistics
print("\nPerforming DBSCAN Clustering Analysis based on Borough Statistics...")

# Aggregate data by borough
print("Aggregating accident data by borough...")

# Use spatial join to determine which borough each accident belongs to
casualty_with_borough = gpd.sjoin(casualty_gdf, london_boroughs, how='left', pre

# Group by borough and calculate statistics
borough_stats = casualty_with_borough.groupby('NAME').agg({
    'accident_index': 'count',  # Total accidents per borough
    'casualty_severity': 'mean'  # Average severity (higher value means more sev
}).reset_index()

# Rename columns for clarity
borough_stats.columns = ['borough_name', 'accident_count', 'avg_severity']

# Filter out any rows with missing data
```

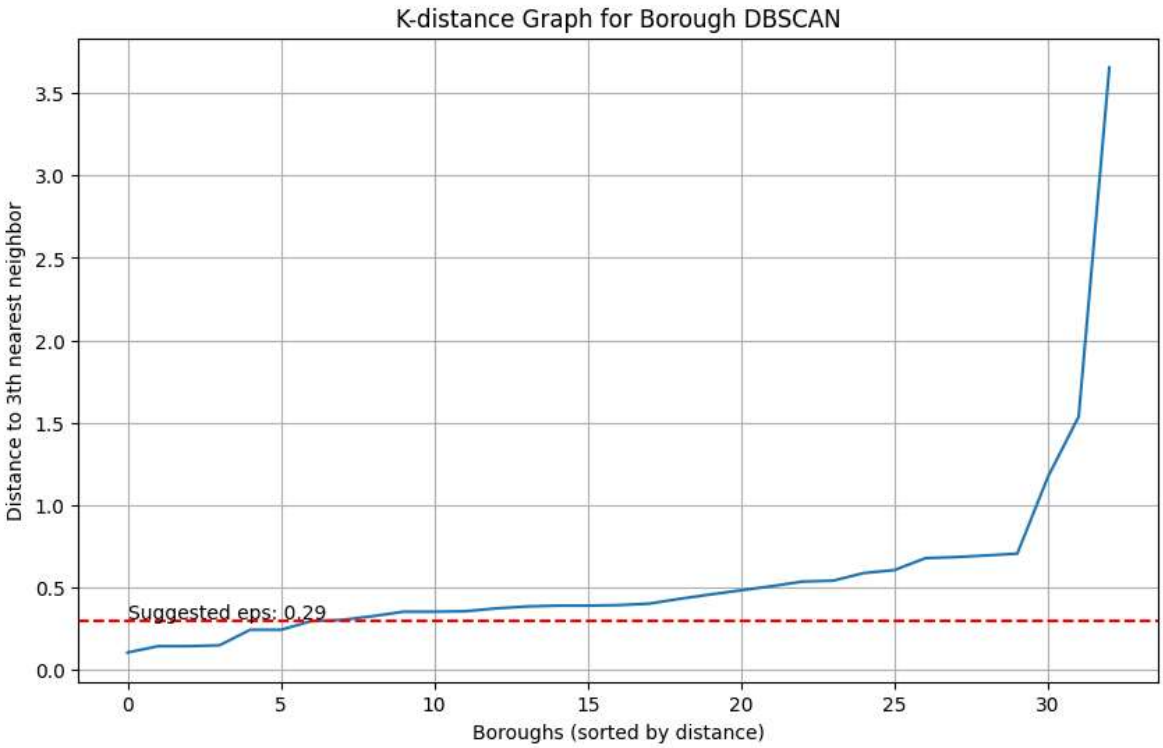
```
borough_stats = borough_stats.dropna()

print("\nBorough statistics summary:")
print(borough_stats.sort_values('accident_count', ascending=False).head())

# Extract features for clustering
X_borough = borough_stats[['accident_count', 'avg_severity']].values

# Normalize features to give them equal importance
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_borough_scaled = scaler.fit_transform(X_borough)
```

```
Performing DBSCAN Clustering Analysis based on Borough Statistics...
Aggregating accident data by borough...

Borough statistics summary:
     borough_name  accident_count  avg_severity
32     Westminster            4433      1.170765
21         Lambeth            3904      1.166240
29   Tower Hamlets            3594      1.133556
27       Southwark            3545      1.153738
9          Enfield            3502      1.105368
```

Now I use **K-distance** and **Elbow method** to find the most suitable value of `eps` and `min_samples`.

In [16]:
```
# Determine optimal epsilon using k-distance graph
print("Finding optimal DBSCAN parameters for borough clustering...")

k_borough = 3  # Smaller k since we have fewer data points (boroughs)
nbrs_borough = NearestNeighbors(n_neighbors=k_borough).fit(X_borough_scaled)
distances_borough, indices_borough = nbrs_borough.kneighbors(X_borough_scaled)

# Sort distances to kth nearest neighbor
k_dist_borough = distances_borough[:, -1]
k_dist_sorted_borough = np.sort(k_dist_borough)

# Plot k-distance graph
plt.figure(figsize=(10, 6))
plt.plot(range(len(k_dist_sorted_borough)), k_dist_sorted_borough)
plt.xlabel('Boroughs (sorted by distance)')
plt.ylabel(f'Distance to {k_borough}th nearest neighbor')
plt.title('K-distance Graph for Borough DBSCAN')
plt.grid(True)

# Add elbow point suggestion
elbow_idx = max(1, int(len(k_dist_sorted_borough) * 0.2))  # Suggest around 20th
elbow_value = k_dist_sorted_borough[elbow_idx]
plt.axhline(y=elbow_value, color='r', linestyle='--')
plt.text(0, elbow_value * 1.05, f'Suggested eps: {elbow_value:.2f}')

plt.show()
```

```
Finding optimal DBSCAN parameters for borough clustering...
```



The plot shows a suggested eps value of about 0.29 for clustering.

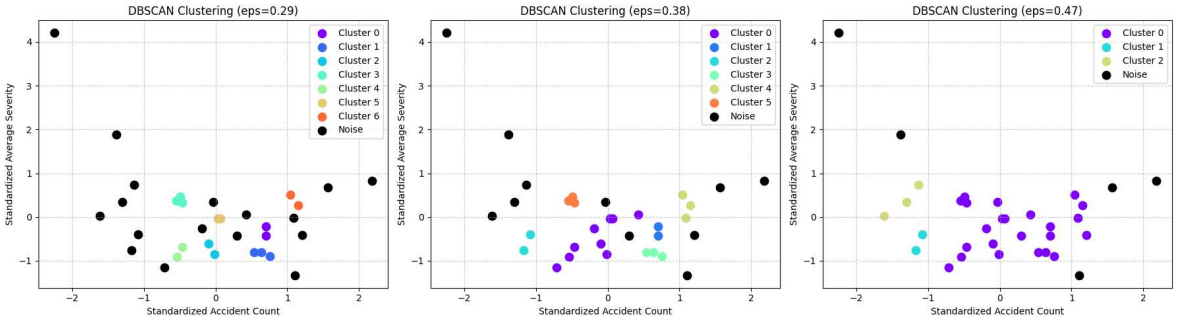
To verify the effect, I **choose different eps values (0.29, 0.38, 0.47) to compare** its cluster counts and noise ratio.

In [17]:
```
# Try different eps values
eps_values = [0.29, 0.38, 0.47]
min_samples = 2

clustering_results = {}

for eps in eps_values:
    # Apply DBSCAN
    dbscan = DBSCAN(eps=eps, min_samples=min_samples)
    cluster_labels = dbscan.fit_predict(X_borough_scaled)

    # Store results
    clustering_results[eps] = {
        'labels': cluster_labels,
        'n_clusters': len(set(cluster_labels)) - (1 if -1 in cluster_labels else
        'n_noise': list(cluster_labels).count(-1)
    }

    # Add cluster labels to borough data
    borough_stats[f'cluster_eps_{eps}'] = cluster_labels

    print(f"eps={eps}: {clustering_results[eps]['n_clusters']} clusters, "
          f"{clustering_results[eps]['n_noise']} noise points "
          f"({clustering_results[eps]['n_noise']/len(cluster_labels):.1%})")

# Evaluate clustering quality using internal metrics
print("\nEvaluating clustering quality...")

from sklearn.metrics import silhouette_score, davies_bouldin_score, calinski_har

# Function to calculate metrics (skipping when only one cluster or all noise)
```

```python
def calculate_metrics(X, labels):
    metrics = {}
    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    n_noise = list(labels).count(-1)

    # Filter out noise points for metric calculation
    if -1 in labels:
        X_filtered = X[labels != -1]
        labels_filtered = labels[labels != -1]
    else:
        X_filtered = X
        labels_filtered = labels

    # Only calculate metrics if there are at least 2 clusters and enough samples
    if n_clusters >= 2 and len(X_filtered) > n_clusters:
        try:
            metrics['silhouette'] = silhouette_score(X_filtered, labels_filtered
            metrics['davies_bouldin'] = davies_bouldin_score(X_filtered, labels_
            metrics['calinski_harabasz'] = calinski_harabasz_score(X_filtered, l
        except:
            pass

    return metrics

# Calculate metrics for each eps value
for eps, result in clustering_results.items():
    labels = result['labels']
    metrics = calculate_metrics(X_borough_scaled, labels)

    print(f"\nCluster quality metrics for eps={eps}:")
    if metrics:
        print(f"  Silhouette Score: {metrics.get('silhouette', 'N/A'):.3f} (high
        print(f"  Davies-Bouldin Index: {metrics.get('davies_bouldin', 'N/A'):.3
        print(f"  Calinski-Harabasz Index: {metrics.get('calinski_harabasz', 'N/
    else:
        print("  Metrics could not be calculated (requires at least 2 clusters w

# Visualize the clustering results in feature space
print("\nVisualizing clustering results in feature space...")

plt.figure(figsize=(18, 5))
for i, eps in enumerate(eps_values):
    plt.subplot(1, len(eps_values), i+1)

    labels = clustering_results[eps]['labels']
    unique_labels = set(labels)

    # Create color map
    colors = plt.cm.rainbow(np.linspace(0, 1, len(unique_labels)))

    # Plot each cluster
    for k, col in zip(unique_labels, colors):
        if k == -1:
            # Black used for noise
            col = 'k'

        mask = (labels == k)
        plt.scatter(X_borough_scaled[mask, 0], X_borough_scaled[mask, 1],
                    c=[col], marker='o', s=80, label=f'Cluster {k}' if k != -1 el
```

```
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.title(f'DBSCAN Clustering (eps={eps})')
    plt.xlabel('Standardized Accident Count')
    plt.ylabel('Standardized Average Severity')
    plt.legend()

plt.tight_layout()
plt.show()
```

```
eps=0.29: 7 clusters, 17 noise points (51.5%)
eps=0.38: 6 clusters, 11 noise points (33.3%)
eps=0.47: 3 clusters, 5 noise points (15.2%)

Evaluating clustering quality...

Cluster quality metrics for eps=0.29:
  Silhouette Score: 0.677 (higher is better, range: -1 to 1)
  Davies-Bouldin Index: 0.364 (lower is better)
  Calinski-Harabasz Index: 85.715 (higher is better)

Cluster quality metrics for eps=0.38:
  Silhouette Score: 0.379 (higher is better, range: -1 to 1)
  Davies-Bouldin Index: 0.626 (lower is better)
  Calinski-Harabasz Index: 14.525 (higher is better)

Cluster quality metrics for eps=0.47:
  Silhouette Score: 0.268 (higher is better, range: -1 to 1)
  Davies-Bouldin Index: 0.639 (lower is better)
  Calinski-Harabasz Index: 9.072 (higher is better)

Visualizing clustering results in feature space...
```



I evaluate each clustering solution using internal metrics:

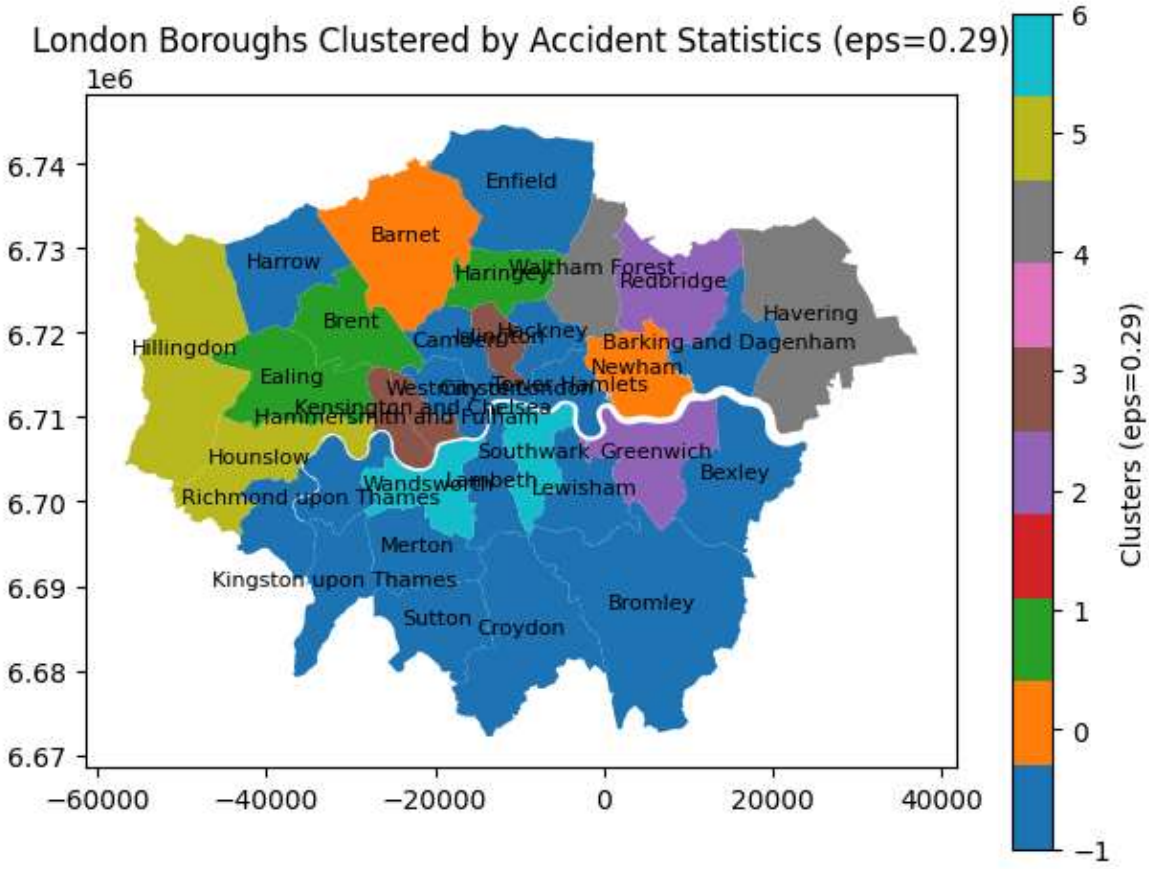
- **Silhouette Score** (higher = better, best at eps=0.29: 0.677)
- **Davies-Bouldin Index** (lower = better, best at eps=0.29: 0.364)
- **Calinski-Harabasz Index** (higher = better, best at eps=0.29: 85.7)

This helps me determine that **eps=0.29 generally produces the best clustering quality**, despite more noise points.

In [18]:
```
# Visualize clusters on London map
print("\nVisualizing clusters on London map...")

# Select one eps value for geographic visualization
selected_eps = 0.29  # Use recommended value or change based on metrics

# Join clustering results back to London boroughs
london_clusters = london_boroughs.copy()
```

```
borough_mapping = dict(zip(borough_stats['borough_name'],
                           borough_stats[f'cluster_eps_{selected_eps}']))
london_clusters['cluster'] = london_clusters['NAME'].map(borough_mapping)

# Create a map for the selected eps value
plt.figure(figsize=(12, 10))
ax = london_clusters.plot(column='cluster', cmap='tab10',
                          legend=True,
                          legend_kwds={'label': f'Clusters (eps={selected_eps})'})

# Add borough names to the map
for idx, row in london_clusters.iterrows():
    plt.annotate(text=row['NAME'], xy=(row.geometry.centroid.x, row.geometry.cen
                ha='center', fontsize=8)

plt.title(f'London Boroughs Clustered by Accident Statistics (eps={selected_eps}
plt.tight_layout()
plt.show()

# Analyze cluster characteristics
print("\nAnalyzing cluster characteristics...")

# For the selected eps, analyze characteristics of each cluster
selected_labels = clustering_results[selected_eps]['labels']
cluster_stats = pd.DataFrame()

for cluster in sorted(set(selected_labels)):
    mask = (selected_labels == cluster)
    cluster_boroughs = borough_stats.loc[mask, 'borough_name'].tolist()

    stats = {
        'Cluster': 'Noise' if cluster == -1 else f'Cluster {cluster}',
        'Number of Boroughs': sum(mask),
        'Boroughs': ', '.join(cluster_boroughs),
        'Avg Accident Count': borough_stats.loc[mask, 'accident_count'].mean(),
        'Min Accident Count': borough_stats.loc[mask, 'accident_count'].min(),
        'Max Accident Count': borough_stats.loc[mask, 'accident_count'].max(),
        'Avg Severity': borough_stats.loc[mask, 'avg_severity'].mean(),
    }

    cluster_stats = pd.concat([cluster_stats, pd.DataFrame([stats])], ignore_ind

print(cluster_stats[['Cluster', 'Number of Boroughs', 'Avg Accident Count', 'Avg
print("\nDetailed borough composition by cluster:")
for _, row in cluster_stats.iterrows():
    print(f"{row['Cluster']}: {row['Boroughs']}")
```

```
Visualizing clusters on London map...
<Figure size 1200x1000 with 0 Axes>
```




```
Analyzing cluster characteristics...
     Cluster  Number of Boroughs  Avg Accident Count  Avg Severity
0      Noise                  17         2393.176471      1.153529
1  Cluster 0                   2         3156.000000      1.135933
2  Cluster 1                   3         3105.000000      1.120587
3  Cluster 2                   2         2498.000000      1.123955
4  Cluster 3                   3         2113.666667      1.157551
5  Cluster 4                   2         2112.000000      1.121866
6  Cluster 5                   2         2590.500000      1.144952
7  Cluster 6                   2         3494.500000      1.157444

Detailed borough composition by cluster:
Noise: Barking and Dagenham, Bexley, Bromley, Camden, City of London, Croydon, En
field, Hackney, Harrow, Kingston upon Thames, Lambeth, Lewisham, Merton, Richmond
upon Thames, Sutton, Tower Hamlets, Westminster
Cluster 0: Barnet, Newham
Cluster 1: Brent, Ealing, Haringey
Cluster 2: Greenwich, Redbridge
Cluster 3: Hammersmith and Fulham, Islington, Kensington and Chelsea
Cluster 4: Havering, Waltham Forest
Cluster 5: Hillingdon, Hounslow
Cluster 6: Southwark, Wandsworth
```

From this map, most boroughs (marked as “Noise” in blue) don't really fit into any clear cluster — they have accident numbers and severity levels that are pretty average or don't match up closely with other areas.

The colored clusters, represent boroughs that stand out. For example, **Cluster 6** includes only a couple of boroughs with the highest average accident counts and severities—they're the real hotspots for accidents.

Some clusters (like **Cluster 4 and Cluster 3**) have lower accident numbers or lower severities, so they might be safer overall.

5.3 Classification analysis

In this section, I will use **Logistic Regression and Random Forests** to analyze traffic behaviors that affect traffic accident severity.

5.3.1 Data preparation and correlation analysis

In [19]:
```
# Copy data for processing
vehicle_analysis = vehicle_df.copy()

# View distribution of dependent variable
print("Accident Severity Distribution:")
print(vehicle_analysis['accident_severity'].value_counts())
print(vehicle_analysis['accident_severity'].value_counts(normalize=True))

# Convert dependent variable to binary classification labels (1:Slight, 2:Seriou
vehicle_analysis['severity_target'] = vehicle_analysis['accident_severity']

# Display data information
print("\nData Information Overview:")
vehicle_analysis.info()
```

```
Accident Severity Distribution:
accident_severity
1    180148
2     60386
Name: count, dtype: int64
accident_severity
1    0.74895
2    0.25105
Name: proportion, dtype: float64

Data Information Overview:
<class 'pandas.core.frame.DataFrame'>
Index: 240534 entries, 21 to 866847
Data columns (total 20 columns):
 #   Column                        Non-Null Count   Dtype
---  ------                        --------------   -----
 0   accident_index                240534 non-null  object
 1   vehicle_reference             240534 non-null  float64
 2   vehicle_type                  240534 non-null  float64
 3   towing_and_articulation       240534 non-null  float64
 4   vehicle_manoeuvre             240534 non-null  float64
 5   vehicle_left_hand_drive       240534 non-null  float64
 6   journey_purpose_of_driver     240534 non-null  float64
 7   sex_of_driver                 240534 non-null  float64
 8   age_of_driver                 240534 non-null  float64
 9   age_band_of_driver            240534 non-null  float64
 10  engine_capacity_cc            240534 non-null  float64
 11  propulsion_code               240534 non-null  float64
 12  age_of_vehicle                240534 non-null  float64
 13  longitude                     240534 non-null  float64
 14  latitude                      240534 non-null  float64
 15  accident_severity             240534 non-null  int64
 16  number_of_vehicles            240534 non-null  int64
 17  number_of_casualties          240534 non-null  int64
 18  local_authority_ons_district  240534 non-null  object
 19  severity_target               240534 non-null  int64
dtypes: float64(14), int64(4), object(2)
memory usage: 38.5+ MB
```

Here I separate the **categorical variables** and the **numerical variables.**

In [20]:
```python
# Define categorical and numerical variables
categorical_features = [
    'vehicle_type', 'towing_and_articulation', 'vehicle_manoeuvre', 'vehicle_lef
    'sex_of_driver', 'propulsion_code'
]

numerical_features = [
    'age_of_driver',  'engine_capacity_cc', 'age_of_vehicle'
]
```

Then, to see which features are actually relevant to accident severity, I test their **correlation** with the target:

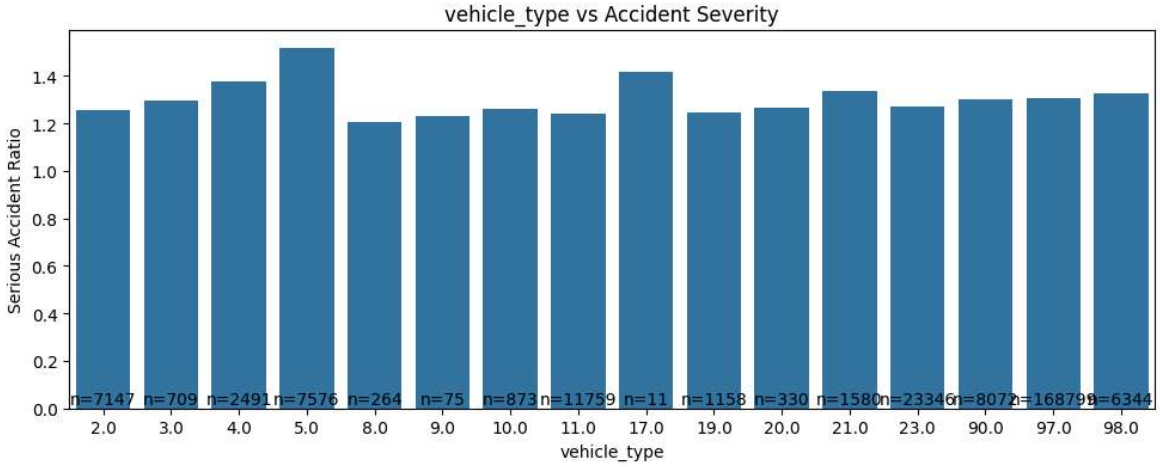
- For categorical variables, I use **chi-square test**;
- For numerical ones, I use **point-biserial correlation.**

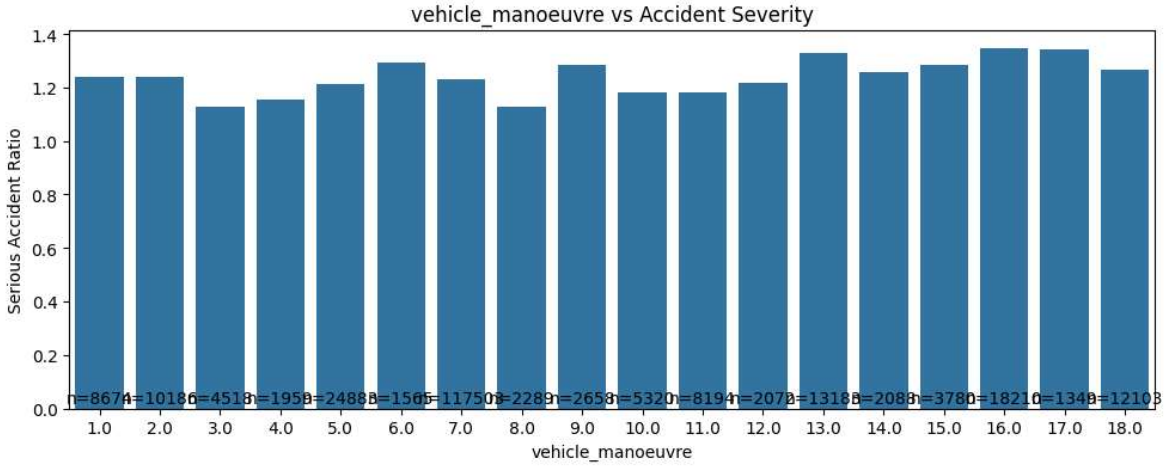
In [21]:
```
# Chi-square test for categorical variables
print("\nAssociation between categorical variables and accident severity (Chi-sq
chi2_results = {}

for feature in categorical_features:
    # Create contingency table
    contingency_table = pd.crosstab(vehicle_analysis[feature], vehicle_analysis[

    # Perform chi-square test
    try:
        chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)
        chi2_results[feature] = {'chi2': chi2_stat, 'p_value': p_value, 'dof': d
        significance = "Significant" if p_value < 0.05 else "Not significant"
        print(f"{feature}: chi2={chi2_stat:.2f}, p={p_value:.4f}, {significance}
    except:
        print(f"{feature}: Unable to calculate chi-square statistic (possibly du

# Point-Biserial Correlation for numerical variables
print("\nCorrelation between numerical variables and accident severity (Point-Bi
pbcorr_results = {}

for feature in numerical_features:
    try:
        # Calculate point-biserial correlation coefficient
        correlation, p_value = pointbiserialr(
            vehicle_analysis[feature],
            vehicle_analysis['severity_target']
        )
        pbcorr_results[feature] = {'correlation': correlation, 'p_value': p_valu
        significance = "Significant" if p_value < 0.05 else "Not significant"
        print(f"{feature}: corr={correlation:.4f}, p={p_value:.4f}, {significanc
    except:
        print(f"{feature}: Unable to calculate point-biserial correlation coeffi
```

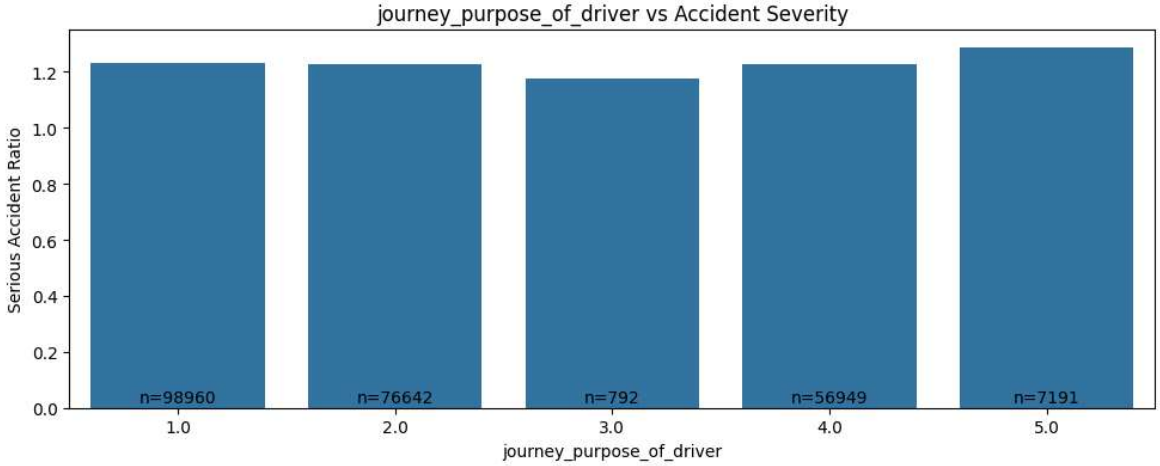
```
Association between categorical variables and accident severity (Chi-square tes
t):
vehicle_type: chi2=3795.58, p=0.0000, Significant
towing_and_articulation: chi2=227.72, p=0.0000, Significant
vehicle_manoeuvre: chi2=3559.93, p=0.0000, Significant
vehicle_left_hand_drive: chi2=0.08, p=0.7833, Not significant
journey_purpose_of_driver: chi2=1164.93, p=0.0000, Significant
sex_of_driver: chi2=726.46, p=0.0000, Significant
propulsion_code: chi2=155.71, p=0.0000, Significant

Correlation between numerical variables and accident severity (Point-Biserial Cor
relation):
age_of_driver: corr=0.0428, p=0.0000, Significant
engine_capacity_cc: corr=0.0236, p=0.0000, Significant
age_of_vehicle: corr=0.0229, p=0.0000, Significant
```

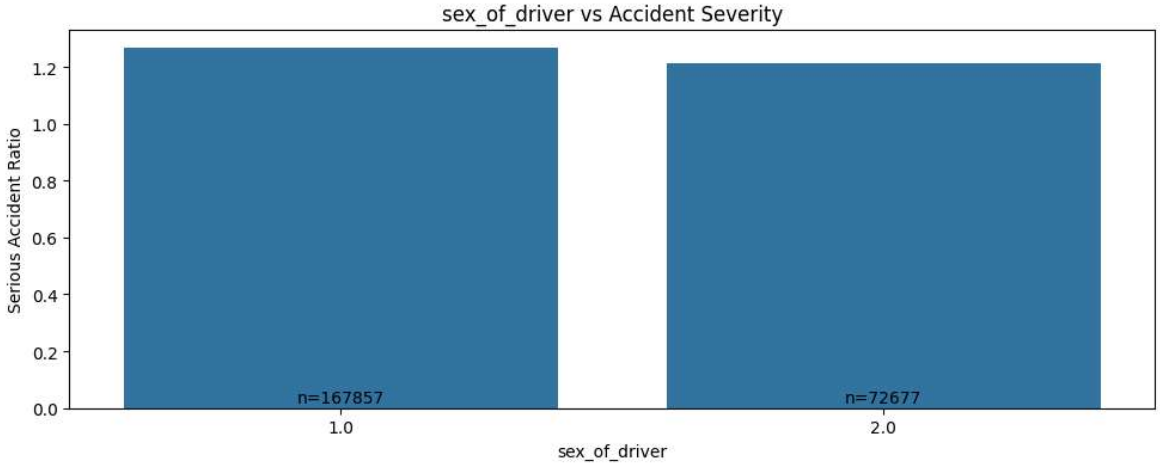
In [22]:
```
# Select top 5 categorical variables with lowest p-values
significant_cat_features = sorted(chi2_results.items(), key=lambda x: x[1]['p_va
significant_cat_names = [item[0] for item in significant_cat_features]

# Plot relationship between categorical variables and accident severity
fig, axes = plt.subplots(len(significant_cat_names), 1, figsize=(10, 4*len(signi

for i, feature in enumerate(significant_cat_names):
    # Calculate the proportion of serious accidents for each category
    severity_by_category = vehicle_analysis.groupby(feature)['severity_target'].
```

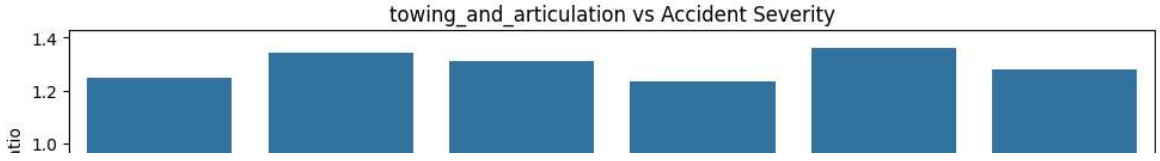
```
    # Calculate sample size for each category
    counts_by_category = vehicle_analysis[feature].value_counts()

    # Create DataFrame for plotting
    plot_df = pd.DataFrame({
        'Category': severity_by_category.index,
        'Severity_Ratio': severity_by_category.values,
        'Count': counts_by_category.reindex(severity_by_category.index).values
    })

    # Sort by severity ratio
    plot_df = plot_df.sort_values('Severity_Ratio', ascending=False)

    # Plot bar chart
    sns.barplot(x='Category', y='Severity_Ratio', data=plot_df, ax=axes[i])
    axes[i].set_title(f'{feature} vs Accident Severity')
    axes[i].set_ylabel('Serious Accident Ratio')
    axes[i].set_xlabel(feature)

    # Add sample size labels
    for j, count in enumerate(plot_df['Count']):
        axes[i].text(j, 0.02, f'n={count}', ha='center')

plt.tight_layout()
plt.savefig('vehicle_categorical_features.png', dpi=300)
plt.show()
```

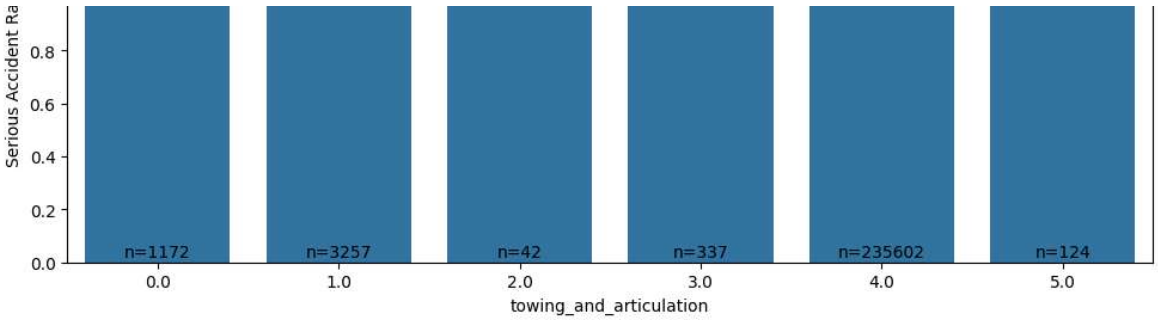

vehicle_type vs Accident Severity
Serious Accident Ratio
vehicle_type
2.0
3.0
4.0
5.0
8.0
9.0
10.0
11.0
17.0
19.0
20.0
21.0
23.0
90.0
97.0
98.0
n=7147
n=709
n=2491
n=7576
n=264
n=75
n=873
n=11759
n=11
n=1158
n=330
n=1580
n=23346
n=6344


vehicle_manoeuvre vs Accident Severity
Serious Accident Ratio
vehicle_manoeuvre
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0


journey_purpose_of_driver vs Accident Severity
Serious Accident Ratio
journey_purpose_of_driver
n=98960
n=76642
n=792
n=56949
n=7191
1.0
2.0
3.0
4.0
5.0


sex_of_driver vs Accident Severity
Serious Accident Ratio
sex_of_driver
n=167857
n=72677
1.0
2.0


towing_and_articulation vs Accident Severity



I keep only the features with p-values lower than 0.05.

Next, I prepare the input features by combining the chosen categorical and numerical variables. I split the dataset into a **training set (80%)** and a **test set (20%).**

To handle missing values and scale the data correctly, I set up preprocessing pipelines:

- For categorical variables, I used **imputation and one-hot encoding.**
- For numerical variables, I used **imputation and standardization.**

In [23]:
```
# Select features for the model
selected_cat_features = [f for f in categorical_features if f in chi2_results an
selected_num_features = [f for f in numerical_features if f in pbcorr_results an

print(f"Selected categorical features: {selected_cat_features}")
print(f"Selected numerical features: {selected_num_features}")

# Prepare features and target variables
X = vehicle_analysis[selected_cat_features + selected_num_features]
y = vehicle_analysis['severity_target']

# 3.1 Train-test split (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

print(f"Training set size: {X_train.shape[0]} samples")
print(f"Test set size: {X_test.shape[0]} samples")

# One-hot encoder for categorical variables
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
])

# Standardizer for numerical variables
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

preprocessor_lr = ColumnTransformer(
    transformers=[
        ('cat', categorical_transformer, selected_cat_features),
        ('num', numerical_transformer, selected_num_features)
```

```
    ]
)

rf_categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent'))
])

preprocessor_rf = ColumnTransformer(
    transformers=[
        ('cat', rf_categorical_transformer, selected_cat_features),
        ('num', numerical_transformer, selected_num_features)
    ]
)
```

```
Selected categorical features: ['vehicle_type', 'towing_and_articulation', 'vehic
le_manoeuvre', 'journey_purpose_of_driver', 'sex_of_driver', 'propulsion_code']
Selected numerical features: ['age_of_driver', 'engine_capacity_cc', 'age_of_vehi
cle']
Training set size: 192427 samples
Test set size: 48107 samples
```

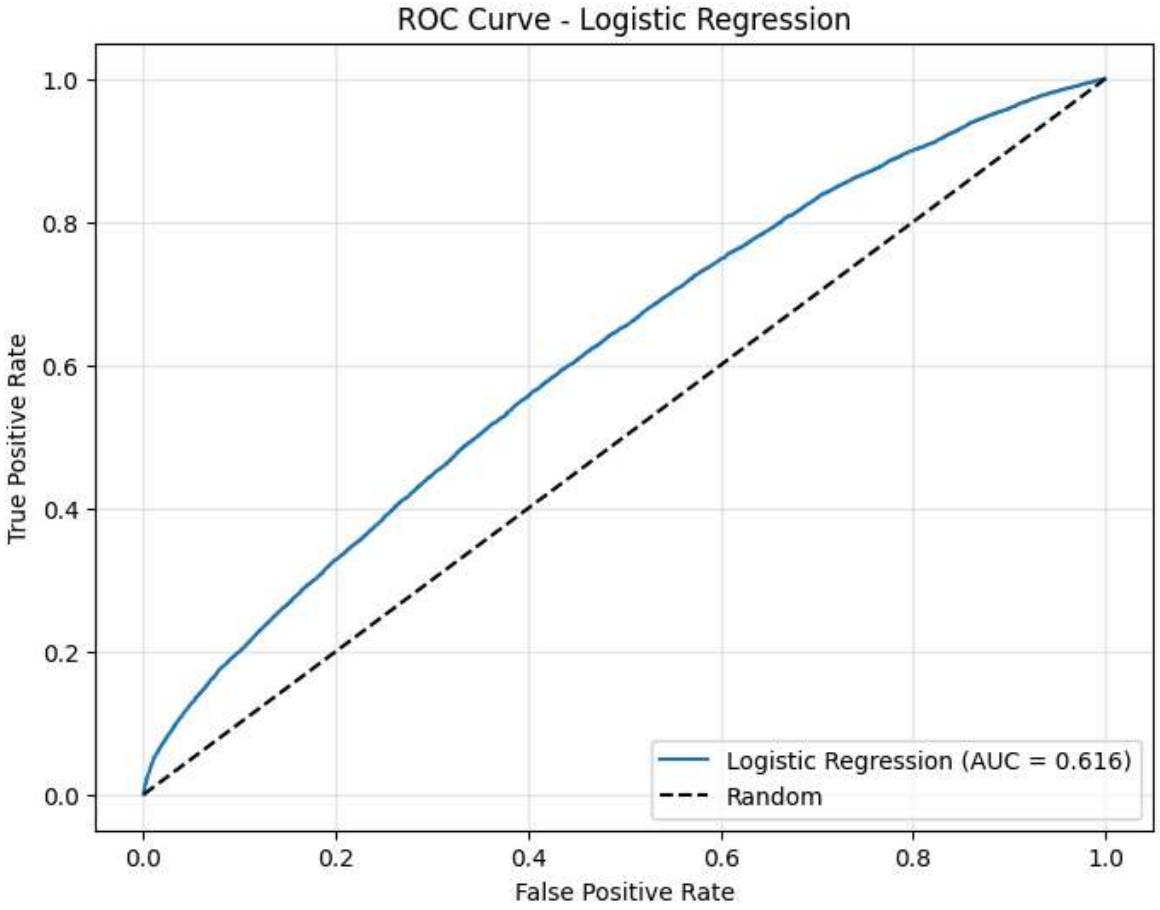
5.3.2 Logistic regression model

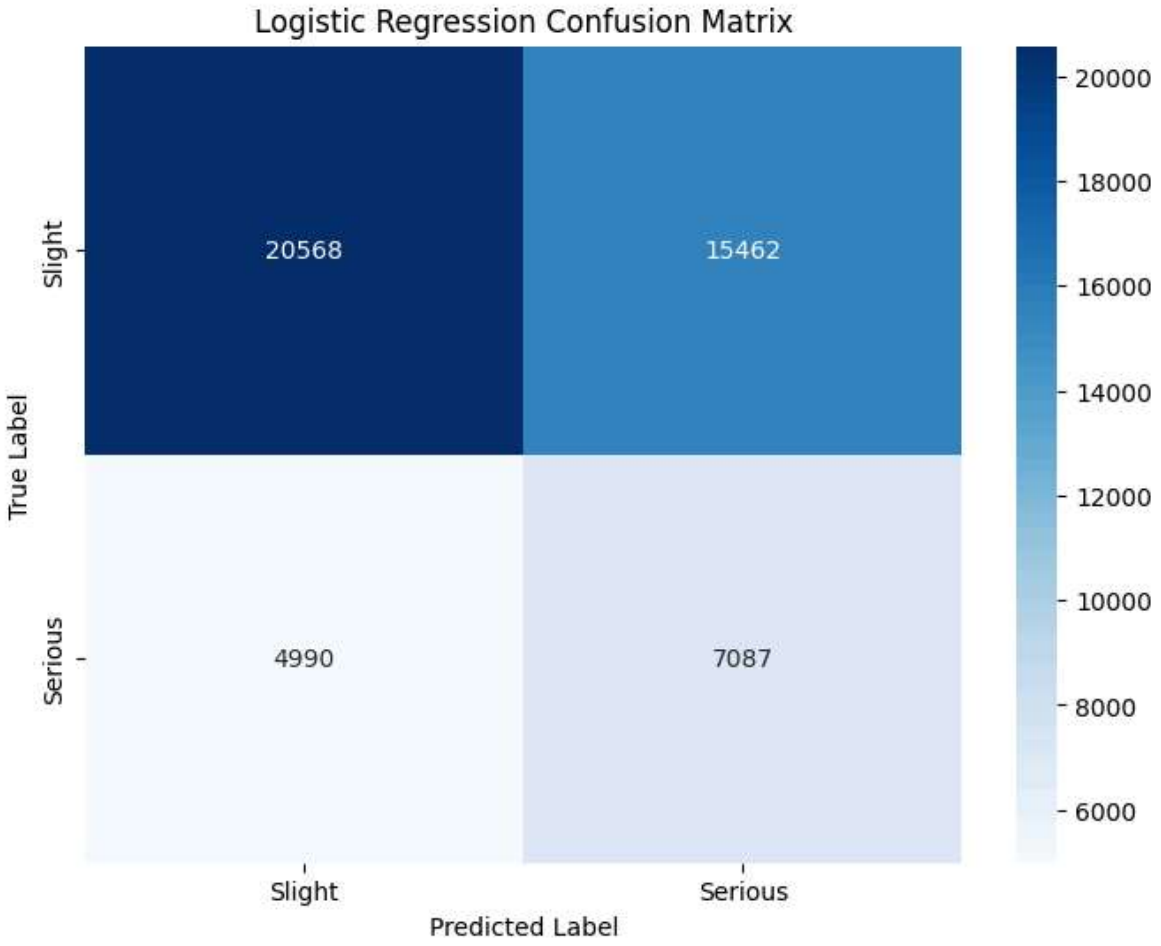
In [24]:

```
# Build logistic regression model
log_reg_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor_lr),
    ('classifier', LogisticRegression(max_iter=1000, random_state=42, class_weig
])

# Train logistic regression model
log_reg_pipeline.fit(X_train, y_train)
y_pred_lr = log_reg_pipeline.predict(X_test)
y_prob_lr = log_reg_pipeline.predict_proba(X_test)[:, 1]

# Evaluate logistic regression model
print("\nLogistic Regression Model Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_lr):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_lr))

print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred_lr)
print(conf_matrix)

# Calculate ROC curve and AUC
fpr_lr, tpr_lr, _ = roc_curve(y_test, y_prob_lr, pos_label=2)
auc_lr = roc_auc_score(y_test, y_prob_lr)
print(f"AUC: {auc_lr:.4f}")

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_lr, tpr_lr, label=f'Logistic Regression (AUC = {auc_lr:.3f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Logistic Regression')
plt.legend(loc='lower right')
plt.grid(True, alpha=0.3)
plt.savefig('logistic_regression_roc.png', dpi=300)
plt.show()
```

```python
# Plot confusion matrix heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Slight', 'Serious'],
            yticklabels=['Slight', 'Serious'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Logistic Regression Confusion Matrix')
plt.savefig('logistic_regression_confusion_matrix.png', dpi=300)
plt.show()
```

```
Logistic Regression Model Evaluation:
Accuracy: 0.5749

Classification Report:
              precision    recall  f1-score   support

           1       0.80      0.57      0.67     36030
           2       0.31      0.59      0.41     12077

    accuracy                           0.57     48107
   macro avg       0.56      0.58      0.54     48107
weighted avg       0.68      0.57      0.60     48107


Confusion Matrix:
[[20568 15462]
 [ 4990  7087]]
AUC: 0.6156
```





The overall accuracy of the model is about **57%.**

From the **confusion matrix**, it's clear the model is better at spotting 'Slight' accidents than 'Serious' ones.

Looking at the classification report, it's harder for the model to catch serious cases, so the F1 score there is lower.

The **ROC curve** shows **AUC=0.62**, which means the model does better than random guessing, but there's still room to improve.

Overall, the model works okay, but it's not great at finding all the serious accidents yet.

5.3.3 Random forest

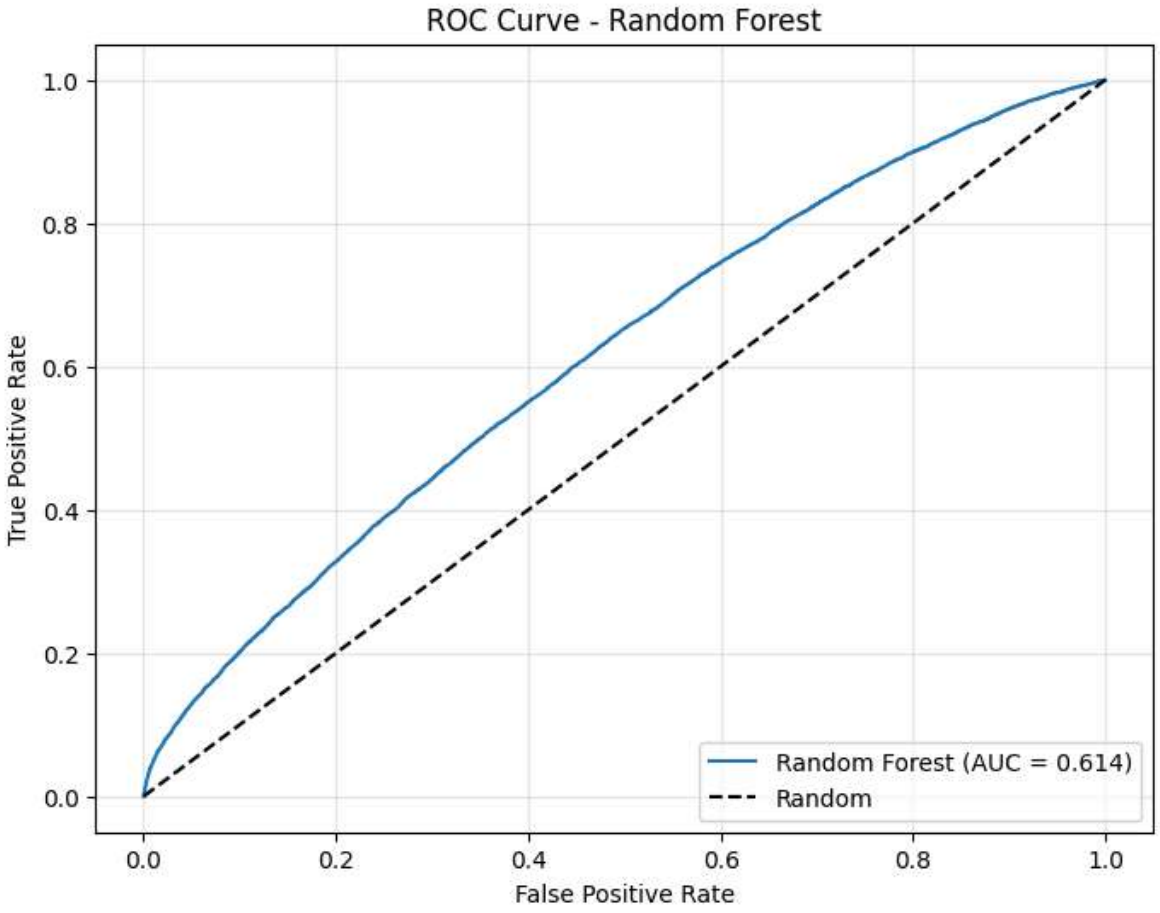
Unlike logistic regression, random forest can automatically handle complex relationships and interactions in the data, and it's less sensitive to outliers.

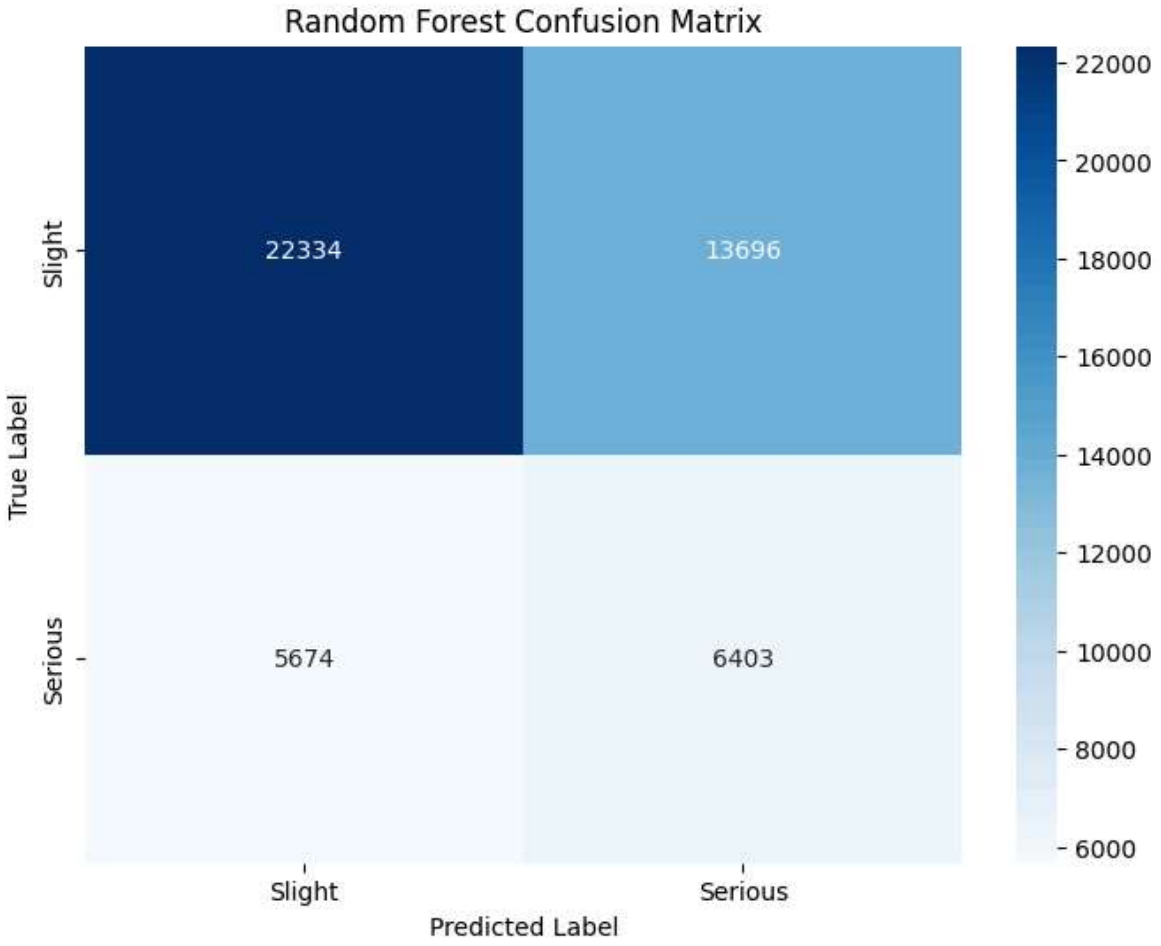
It's also helpful for dealing with a mix of numerical and categorical features.

In [25]:
```
# Build random forest model
rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor_rf),
    ('classifier', RandomForestClassifier(
        n_estimators=100,
        max_depth=10,
```

```
        min_samples_split=5,
        min_samples_leaf=2,
        random_state=42,
        class_weight='balanced'
    ))
])

# Train random forest model
rf_pipeline.fit(X_train, y_train)
y_pred_rf = rf_pipeline.predict(X_test)
y_prob_rf = rf_pipeline.predict_proba(X_test)[:, 1]

# Evaluate random forest model
print("\nRandom Forest Model Evaluation:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf):.4f}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred_rf))

print("\nConfusion Matrix:")
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
print(conf_matrix_rf)

# Calculate ROC curve and AUC
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_prob_rf, pos_label=2)
auc_rf = roc_auc_score(y_test, y_prob_rf)
print(f"AUC: {auc_rf:.4f}")

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {auc_rf:.3f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve - Random Forest')
plt.legend(loc='lower right')
plt.grid(True, alpha=0.3)
plt.savefig('random_forest_roc.png', dpi=300)
plt.show()

# Plot confusion matrix heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Slight', 'Serious'],
            yticklabels=['Slight', 'Serious'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Random Forest Confusion Matrix')
plt.savefig('random_forest_confusion_matrix.png', dpi=300)
plt.show()
```

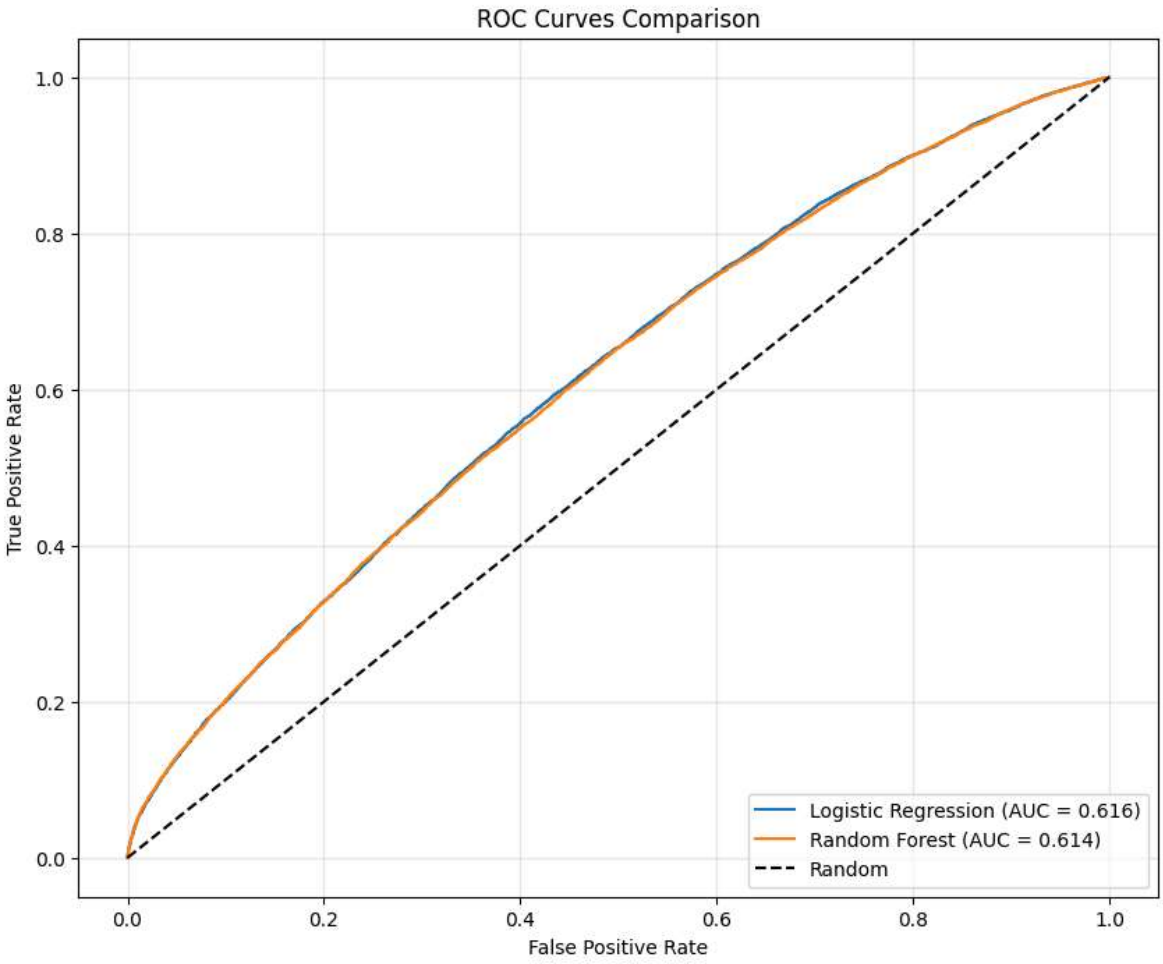
```
Random Forest Model Evaluation:
Accuracy: 0.5974

Classification Report:
              precision    recall  f1-score   support

           1       0.80      0.62      0.70     36030
           2       0.32      0.53      0.40     12077

    accuracy                           0.60     48107
   macro avg       0.56      0.58      0.55     48107
weighted avg       0.68      0.60      0.62     48107


Confusion Matrix:
[[22334 13696]
 [ 5674  6403]]
AUC: 0.6137
```





After training and testing both logistic regression and random forest models, I compare their results to see if using a more advanced algorithm would make a big difference.

In [26]:
```
# Compare ROC curves of both models
plt.figure(figsize=(10, 8))
plt.plot(fpr_lr, tpr_lr, label=f'Logistic Regression (AUC = {auc_lr:.3f})')
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {auc_rf:.3f})')
plt.plot([0, 1], [0, 1], 'k--', label='Random')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves Comparison')
plt.legend(loc='lower right')
plt.grid(True, alpha=0.3)
plt.savefig('models_comparison_roc.png', dpi=300)
plt.show()
```



Looking at the ROC curves and AUC scores, **both models performed similarly**.

The random forest model achieves an AUC=0.614, which is even slightly lower than logistic regression.

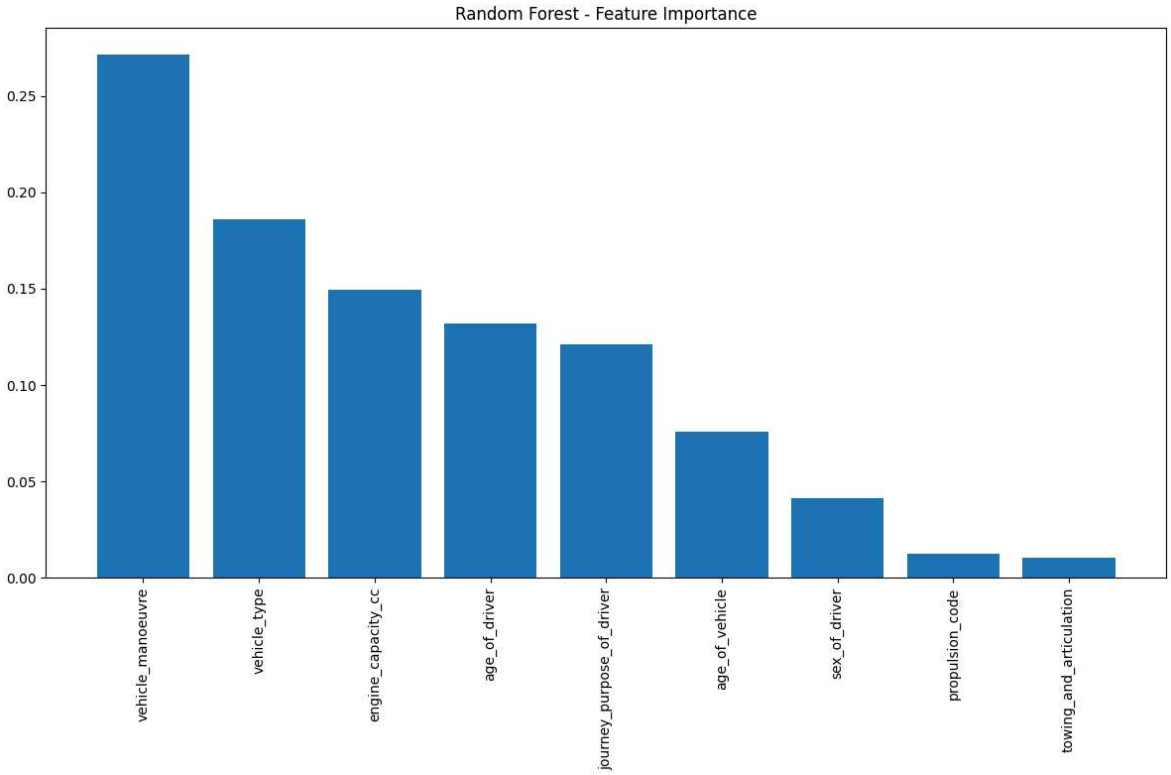
The main reason these two models have similar performance is likely that:

- The features don't provide enough strong signals to separate the two accident severity classes very clearly.

In summary, although random forest is a more advanced model and is expected to do better, in this case, both models ende up with close results because the underlying data patterns are not very distinct.

In [27]:
```
# Calculate and display random forest feature importance
if selected_cat_features or selected_num_features:
    # Get random forest model
    rf_model = rf_pipeline.named_steps['classifier']

    # Since we're not using one-hot encoding for random forest,
    # the feature names are simply the original feature names
    feature_names = selected_cat_features + selected_num_features

    # Get feature importance
    importances = rf_model.feature_importances_
    indices = np.argsort(importances)[::-1]

    # Only plot top features
```

```
    n_features_to_plot = min(len(feature_names), len(importances))

    plt.figure(figsize=(12, 8))
    plt.title('Random Forest - Feature Importance')
    plt.bar(range(n_features_to_plot),
            [importances[i] for i in indices[:n_features_to_plot]],
            align='center')
    plt.xticks(range(n_features_to_plot),
               [feature_names[i] if i < len(feature_names) else str(i) for i in
               rotation=90)
    plt.tight_layout()
    plt.show()
```



From the feature importance chart, the top 3 features that have a greater impact on the severity of traffic accidents are:

- vehicle_manoeuvre
- vehicle_type
- engine_capacity_cc

6.Conclusion

[go back to the top]

- Some London boroughs have more frequent and more severe accidents. Places like Hammersmith and Fulham, Islington, Kensington and Chelsea really stand out, while most other areas are pretty average.
- Both logistic regression and random forest models show similar results. This is probably because the data doesn't have really strong patterns for predicting severity.

- The Top 3 factors for making an accident more serious are vehicle_manoeuvre, vehicle_type and engine_capacity_cc.

References

[go back to the top]

Castro, Y. and Kim, Y. J. (2015) 'Data mining on road safety: factor assessment on vehicle accidents using classification models', International Journal of Crashworthiness, 21(2), pp. 104-111. doi: 10.1080/13588265.2015.1122278.

Choudhary, J. K., Rayala, N., Kiasari, A. E. and Jafari, F. (2023). 'Road Safety in Great Britain: An Exploratory Data Analysis'. International Journal of Transport and Vehicle Engineering. World Academy of Science, Engineering and Technology, 17 (7), pp. 273-287.

Rebecca C. Gray, Mohammed A. Quddus, Andrew Evans(2008). 'Injury severity analysis of accidents involving young male drivers in Great Britain'. Journal of Safety Research. Pergamon, 39 (5), pp. 483-495. doi: 10.1016/j.jsr.2008.07.003.

Marchant, P., Hale, J. D. and Sadler, J. P. (2020). 'Does changing to brighter road lighting improve road safety? Multilevel longitudinal analysis of road traffic collision frequency during the relighting of a UK city'. J Epidemiol Community Health. BMJ Publishing Group Ltd, 74 (5), pp. 467-472. doi: 10.1136/jech-2019-212208.

Salehian, A., Aghabayk, K., Seyfi, M. and Shiwakoti, N. (2023). 'Comparative analysis of pedestrian crash severity at United Kingdom rural road intersections and Non-Intersections using latent class clustering and ordered probit model'. Accident Analysis & Prevention, 192, p. 107231. doi: 10.1016/j.aap.2023.107231.

Zhu, Y. et al. (2024) 'Young novice drivers' road crash injuries and contributing factors: A crash data investigation', Traffic Injury Prevention, 25(8), pp. 1031-1038. doi: 10.1080/15389588.2024.2367504.