submission_final

April 20, 2025

1 Title: Severity of road traffic accidents

[1]:
```
%load_ext watermark
%watermark -a "Van Wu" -u -d -t -v -p numpy,pandas,matplotlib,scikit-learn
```

```
Author: Van Wu

Last updated: 2025-04-19 22:14:29

Python implementation: CPython
Python version       : 3.13.2
IPython version      : 9.0.2

numpy       : 2.2.4
pandas      : 2.2.3
matplotlib  : 3.10.1
scikit-learn: 1.6.1
```

1.1 Preparation

- Github link
- Number of words: ***
- Runtime: *** hours (*Memory 32 GB, CPU AMD Ryzen 7 5800H with Radeon Graphics CPU @3.20GHz*)
- Coding environment: Coding environment: VS Code with Jupyter plugin (local), not SDS Docker
- License: this notebook is made available under the Creative Commons Attribution license.
- Additional library *[libraries not included in SDS Docker or not used in this module]*:
  - **watermark**: A Jupyter Notebook extension for printing timestamps, version numbers, and hardware information.(used to print Python and package versions for reproducibility.)
  - ……

1.2 Table of contents

1.3 Introduction

[go back to the top]

Road traffic accidents (RTAs) represent a significant public health and urban governance issue globally. In the UK, despite advancements in vehicle technology and traffic regulation, thousands of individuals are injured or killed on the roads annually. Predicting the severity of these accidents is crucial for targeted policy interventions and infrastructure planning. Accident severity is influenced by a range of contextual factors including weather, road geometry, traffic volume, time of day, and infrastructure design (Abdel-Aty & Haleem, 2011). As cities move towards data-driven governance, the use of machine learning models has become increasingly common in road safety research (Zhang et al., 2020).

Recent literature has demonstrated the effectiveness of supervised learning algorithms such as logistic regression, random forests, and XGBoost in predicting accident severity using structured datasets (Ahmed et al., 2023). These models are particularly suitable for capturing non-linear interactions and heterogeneous effects among multiple explanatory variables. Moreover, explainable AI techniques such as SHAP (SHapley Additive exPlanations) have been widely adopted to interpret complex models and understand feature importance, which aids in translating statistical findings into actionable insights for policymakers.

This study leverages the UK Department for Transport's Road Safety Data (2015–2019), which documents detailed information on individual accident cases, including temporal, spatial, environmental, and infrastructural attributes. By integrating network-based features such as road betweenness centrality extracted via OpenStreetMap, this project attempts to bridge the gap between spatial network analysis and predictive modelling of accident severity. The objective is twofold: to evaluate the predictive performance of commonly used machine learning models on accident severity classification, and to examine the relative contribution of different spatial and contextual factors to the outcome.

The period from 2015 to 2019 was deliberately chosen to ensure data stability and validity. This timeframe avoids the confounding effects of the COVID-19 pandemic (2020–2021), which significantly disrupted travel behaviour, enforcement levels, and urban mobility patterns across the UK (DfT, 2021). It also precedes Phase 2 of London's major road transformation programme, including the expansion of Low Traffic Neighbourhoods (LTNs) and segregated cycling infrastructure, which introduced substantial structural changes to the transport system from 2020 onwards (TfL, 2024). In contrast, the preceding years (2010–2014) marked a foundational policy phase, characterised by the implementation of new traffic enforcement measures such as fixed penalty notices for careless driving and increased fines for common violations (DfT, 2013). By focusing on the relatively stable and mature period between 2015 and 2019, this study ensures greater internal consistency and enables clearer interpretation of accident severity patterns, isolated from exogenous policy or behavioural shocks.

[ ]:

1.4 Research questions

[go back to the top]

Can supervised machine learning models accurately predict the severity of road traffic accidents in London using spatial, temporal, and environmental features?

This study investigates whether supervised machine learning models can accurately predict the severity of road traffic accidents in London based on spatial, temporal, and environmental features. Specifically, it examines the predictive power of variables such as time of day, weather conditions, and road network centrality. The study also compares the performance of Logistic Regression, Random Forest, and XGBoost, and uses SHAP analysis to identify the most influential features for each severity level (fatal, serious, slight).

[ ]:

1.5 Data

[go back to the top]

1.5.1 1. Data Description

Variable	Type	Description	Notes
accident_severity	Categorical	Severity level of the accident (1 = Fatal, 2 = Serious, 3 = Slight)	Target variable
speed_limit	Numeric	Speed limit of the road segment (in mph)	-
accident_year	Numeric	Year of the accident (2015–2019)	Used for train-test split
mean_betweenness	Numeric	Mean betweenness centrality of nearby road segments	Spatial network feature
max_betweenness	Numeric	Maximum betweenness centrality of nearby road segments	Key spatial variable
mean_degree	Numeric	Mean degree centrality of road network	-
max_degree	Numeric	Maximum degree centrality	-
edge_count	Numeric	Number of road segments (edges) in the local road network	Spatial indicator of network density
day_of_week_*	Categorical	One-hot encoded day of week (Monday–Saturday, Sunday as baseline)	One-hot encoded
road_type_*	Categorical	One-hot encoded road type categories	One-hot encoded
light_conditions_*	Categorical	One-hot encoded lighting conditions (e.g., daylight, darkness with/without lighting)	One-hot encoded

Variable	Type	Description	Notes
weather_conditions_*	Categorical	One-hot encoded weather conditions (e.g., fine, rain, fog)	One-hot encoded
road_surface_conditions_*	Categorical	One-hot encoded surface conditions (e.g., dry, wet)	One-hot encoded
junction_control_*	Categorical	One-hot encoded control types at junctions	One-hot encoded
junction_detail_*	Categorical	One-hot encoded structural junction types	One-hot encoded
pedestrian_crossing_human_control_*	Categorical	One-hot encoded presence of human-controlled crossings	One-hot encoded
pedestrian_crossing_physical_facilities_*	Categorical	One-hot encoded presence of physical pedestrian facilities	One-hot encoded
special_conditions_at_site_*	Categorical	One-hot encoded site-specific conditions (e.g., roadworks)	One-hot encoded
first_road_class_*	Categorical	One-hot encoded classification of the primary road	One-hot encoded
second_road_class_*	Categorical	One-hot encoded classification of the secondary road	One-hot encoded
trunk_road_flag_*	Categorical	One-hot encoded trunk road indicator	One-hot encoded
urban_or_rural_area_*	Categorical	One-hot encoded urban/rural area classification	One-hot encoded
time_hour	Numeric	Hour of the accident (e.g., 13:55 → 13)	Derived feature
betweenness_level_encoded	Ordinal	Quartile level of mean_betweenness (0 = Low, 3 = High)	For logistic regression compatibility
……	……	……	

Note: `*_` *denotes one-hot encoded categories split into multiple columns.*

The following table provides code-level descriptions for categorical variables used in this study. Definitions are based on the official UK Department for Transport data guide: data.gov.uk.

Variable Prefix	Code	Meaning
day_of_week	1	Sunday
	2	Monday
	3	Tuesday
	4	Wednesday
	5	Thursday
	6	Friday
	7	Saturday
road_type	1	Roundabout
	2	One way street
	3	Dual carriageway
	6	Single carriageway
	7	Slip road
	9	Unknown
light_conditions	1	Daylight
	4	Darkness - lights lit

Variable Prefix	Code	Meaning
	5	Darkness - lights unlit
	6	Darkness - no lighting
	7	Darkness - lighting unknown
weather_conditions	1	Fine no high winds
	2	Raining no high winds
	3	Snowing no high winds
	4	Fine + high winds
	5	Raining + high winds
	6	Snowing + high winds
	7	Fog or mist
	8	Other
	9	Unknown
road_surface_conditions	1	Dry
	2	Wet or damp
	3	Snow
	4	Frost or ice
	5	Flood (surface water)
	9	Unknown
junction_control	0	None
	1	Authorised person
	2	Auto traffic signal
	3	Stop sign
	4	Give way or uncontrolled
	9	Unknown
pedestrian_crossing_human_control	0	None
	1	School crossing patrol
	2	Other human control
	9	Unknown
pedestrian_crossing_physical_facilities	0	None
	1	Zebra crossing
	4	Pelican crossing
	5	Footbridge or subway
	7	Refuge
	8	Unknown
	9	Other
urban_or_rural_area	1	Not used
	2	Urban
	3	Rural
trunk_road_flag	1	Non-trunk road
	2	Trunk road
first_road_class / second_road_class	1	Motorway
	2	A(M) Road
	3	A Road
	4	B Road
	5	C Road

1.5.2 2. Data Import and Prepration

[2]:
```
# It would import the packages that would be used first.
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
import os
import osmnx as ox
import networkx as nx
import geopandas as gpd
from tqdm import tqdm
```

[3]:
```
#
input_folder = '../data/raw'
output_folder = '../data/clean'
```

[4]:
```
# Road Data
df = pd.read_csv('../data/raw/1979-latest-published-year.csv')

df = df[df['accident_year'].isin([2015, 2016, 2017, 2018, 2019])]
print(f"2015-2019     {len(df)}  ")

#
df.to_csv("../data/raw/2015_2019.csv", index=False)
```

```
C:\Users\Lenovo\AppData\Local\Temp\ipykernel_25084\3003610021.py:2:
DtypeWarning: Columns (0,2,15,16,35) have mixed types. Specify dtype option on
import or set low_memory=False.
  df = pd.read_csv('../data/raw/1979-latest-published-year.csv')
```

```
2015-2019     646830
```

[5]:
```
columns_to_keep = [
    'accident_severity',
    'number_of_vehicles',
    'number_of_casualties',
    'day_of_week',
    'time',
    'first_road_class',
    'second_road_class',
    'road_type',
    'speed_limit',
    'junction_detail',
    'junction_control',
    'pedestrian_crossing_human_control',
```

```
    'pedestrian_crossing_physical_facilities',
    'light_conditions',
    'weather_conditions',
    'road_surface_conditions',
    'special_conditions_at_site',
    'carriageway_hazards',
    'urban_or_rural_area',
    'did_police_officer_attend_scene_of_accident',
    'trunk_road_flag',
    'local_authority_ons_district',
    'accident_year'
]

selected_columns = [col for col in columns_to_keep if col in df.columns]
df_cleaned = df[selected_columns]

#
missing_counts = df_cleaned.isnull().sum()
total_missing = missing_counts.sum()

if total_missing > 0:
    print(f"    {total_missing}    ")
    print(missing_counts[missing_counts > 0])

    #
    df_cleaned = df_cleaned.dropna()
    print(f"       {len(df_cleaned)}    ")

#
df_cleaned.to_csv('../data/clean/1519_cleaned.csv', index=False)
print(f"       {output_folder}  {len(df_cleaned.columns)}    {len(df_cleaned)}    ")
```

```
    37
speed_limit    37
dtype: int64
       646793
       ../data/clean  23   646793
```

[ ]:

```
# RoadCentrality
path = "../data/Borough_Boundaries.geojson"
boroughs = gpd.read_file(path)
boroughs = boroughs[["name", "gss_code", "geometry"]].rename(columns={"name": "borough"})

ox.settings.log_console = False
ox.settings.use_cache = True
```

```python
results = []

for idx, row in tqdm(boroughs.iterrows(), total=len(boroughs), desc="Processing boroughs"):
    borough_name = row["borough"]
    gss_name = row["gss_code"]
    geometry = row["geometry"]

    try:
        print(f"Processing: {borough_name}")

        G = ox.graph_from_polygon(geometry, network_type="drive", simplify=True)

        betweenness = nx.betweenness_centrality(G, weight="length", k=100, seed=42)
        degree = dict(G.degree())
        nx.set_node_attributes(G, betweenness, "betweenness")
        nx.set_node_attributes(G, degree, "degree")

        edge_data = []
        for u, v, key, data in G.edges(keys=True, data=True):
            edge_data.append({
                "u": u,
                "v": v,
                "key": key,
                "geometry": data.get("geometry", None),
                "betweenness": (G.nodes[u]["betweenness"] + G.nodes[v]["betweenness"]) / 2,
                "degree": (G.nodes[u]["degree"] + G.nodes[v]["degree"]) / 2
            })
        edges_df = gpd.GeoDataFrame(edge_data, geometry="geometry", crs="EPSG:4326")

        summary = {
            "borough": borough_name,
            "gss_code": gss_name,
            "mean_betweenness": edges_df["betweenness"].mean(),
            "max_betweenness": edges_df["betweenness"].max(),
            "mean_degree": edges_df["degree"].mean(),
            "max_degree": edges_df["degree"].max(),
            "edge_count": len(edges_df)
        }
        results.append(summary)

    except Exception as e:
        print(f"  Failed for {borough_name}: {e}")
```

```
        continue

df_results = pd.DataFrame(results)
df_results.to_csv("../data/london_borough_road_centrality.csv", index=False)
print("All done! Results saved to 'london_borough_road_centrality.csv'")
```

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

#
accident_path = "../data/clean/1519_cleaned.csv"
centrality_path = "../data/london_borough_road_centrality.csv"
output_path = "../data/final/2015_2019_with_centrality.csv"

#
df_accident = pd.read_csv(accident_path)
df_centrality = pd.read_csv(centrality_path)

#
df_merged = df_accident.merge(
    df_centrality,
    how="left",
    left_on="local_authority_ons_district",
    right_on="gss_code"
)

#
before_drop = len(df_merged)
df_merged = df_merged.dropna(subset=["mean_betweenness"])
after_drop = len(df_merged)
dropped = before_drop - after_drop

#
df_merged.to_csv(output_path, index=False)

print(f"         {output_path}")
print(f"   {after_drop}     {dropped}       ")
```

```
         ../data/final/2015_2019_with_centrality.csv
  128261     518532
```

1.5.3 3. Data Pattern Analysis

[7]:
```
df = pd.read_csv("../data/final/2015_2019_with_centrality.csv")
```

[8]:
```
print(df.shape)
print(df.dtypes)
```

```python
print(df.isnull().sum())   #
df.describe()              #
df["accident_severity"].value_counts(normalize=True)  #
```

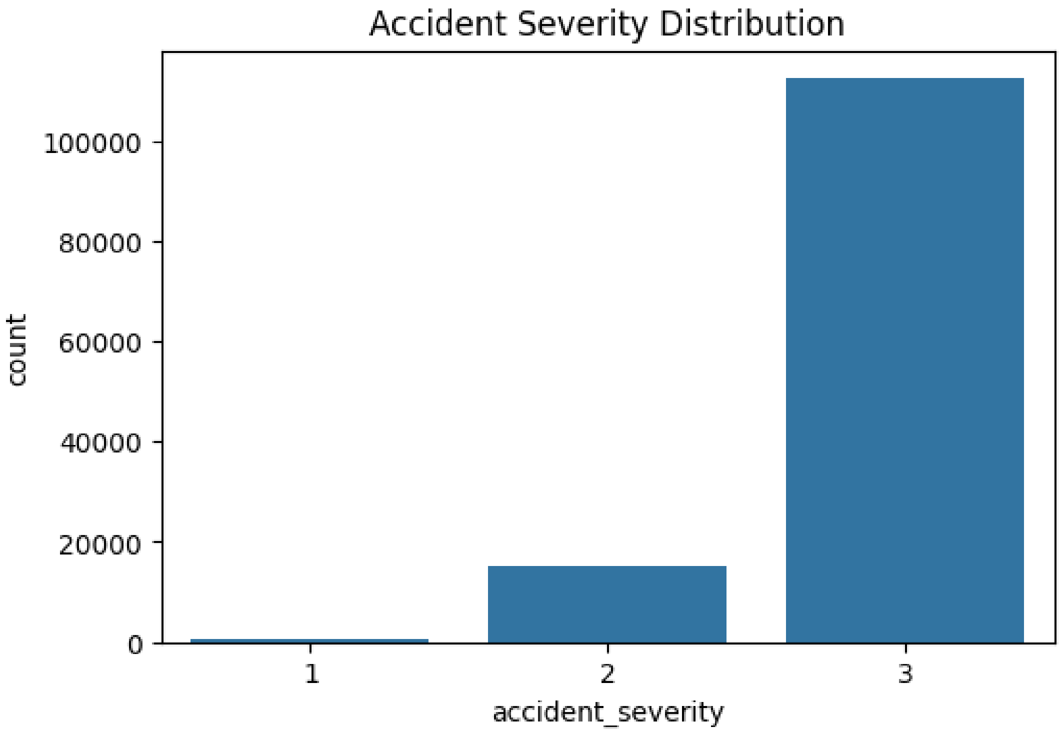
```
(128261, 30)
accident_severity                              int64
number_of_vehicles                             int64
number_of_casualties                           int64
day_of_week                                    int64
time                                          object
first_road_class                               int64
second_road_class                              int64
road_type                                      int64
speed_limit                                  float64
junction_detail                                int64
junction_control                               int64
pedestrian_crossing_human_control              int64
pedestrian_crossing_physical_facilities        int64
light_conditions                               int64
weather_conditions                             int64
road_surface_conditions                        int64
special_conditions_at_site                     int64
carriageway_hazards                            int64
urban_or_rural_area                            int64
did_police_officer_attend_scene_of_accident    int64
trunk_road_flag                                int64
local_authority_ons_district                  object
accident_year                                  int64
borough                                       object
gss_code                                      object
mean_betweenness                             float64
max_betweenness                              float64
mean_degree                                  float64
max_degree                                   float64
edge_count                                   float64
dtype: object
accident_severity                            0
number_of_vehicles                           0
number_of_casualties                         0
day_of_week                                  0
time                                         0
first_road_class                             0
second_road_class                            0
road_type                                    0
speed_limit                                  0
junction_detail                              0
junction_control                             0
pedestrian_crossing_human_control            0
```

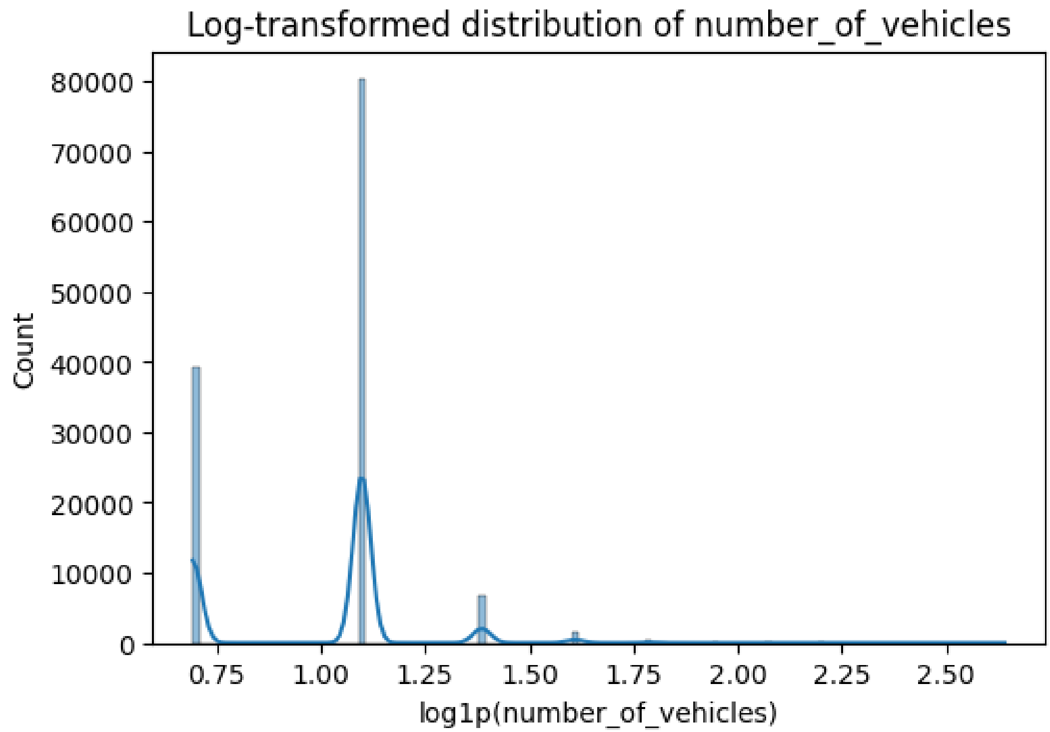
```
pedestrian_crossing_physical_facilities        0
light_conditions                               0
weather_conditions                             0
road_surface_conditions                        0
special_conditions_at_site                     0
carriageway_hazards                            0
urban_or_rural_area                            0
did_police_officer_attend_scene_of_accident    0
trunk_road_flag                                0
local_authority_ons_district                   0
accident_year                                  0
borough                                        0
gss_code                                       0
mean_betweenness                               0
max_betweenness                                0
mean_degree                                    0
max_degree                                     0
edge_count                                     0
dtype: int64
```

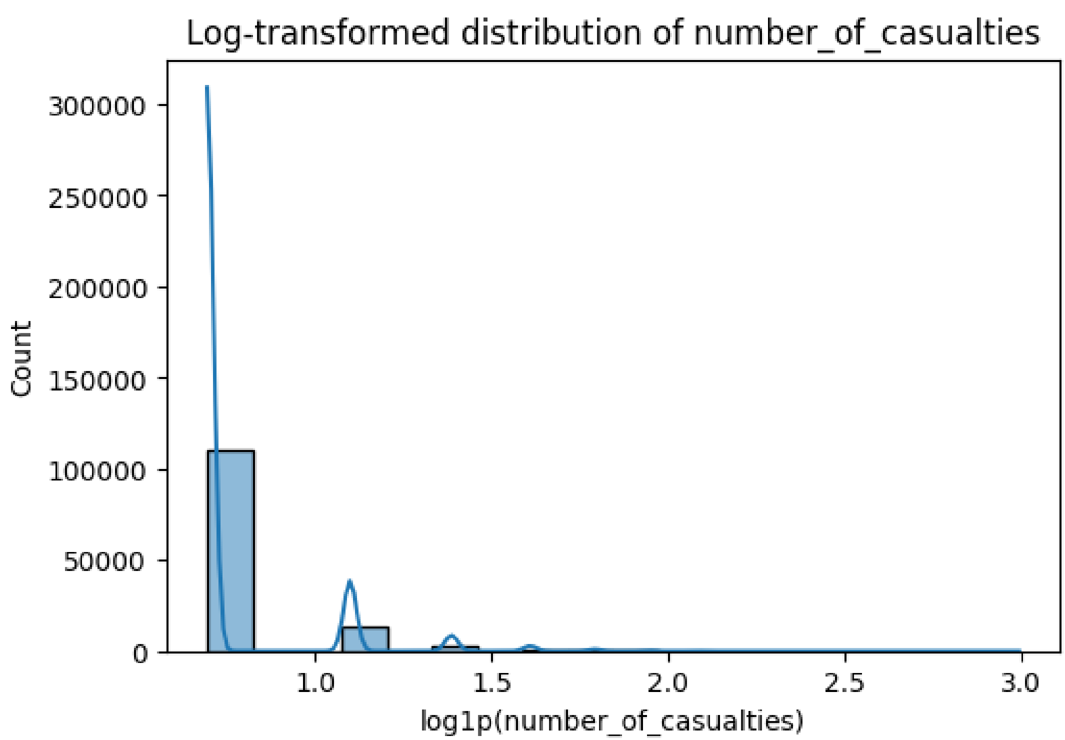
[8]:
```
accident_severity
3    0.876089
2    0.119179
1    0.004733
Name: proportion, dtype: float64
```

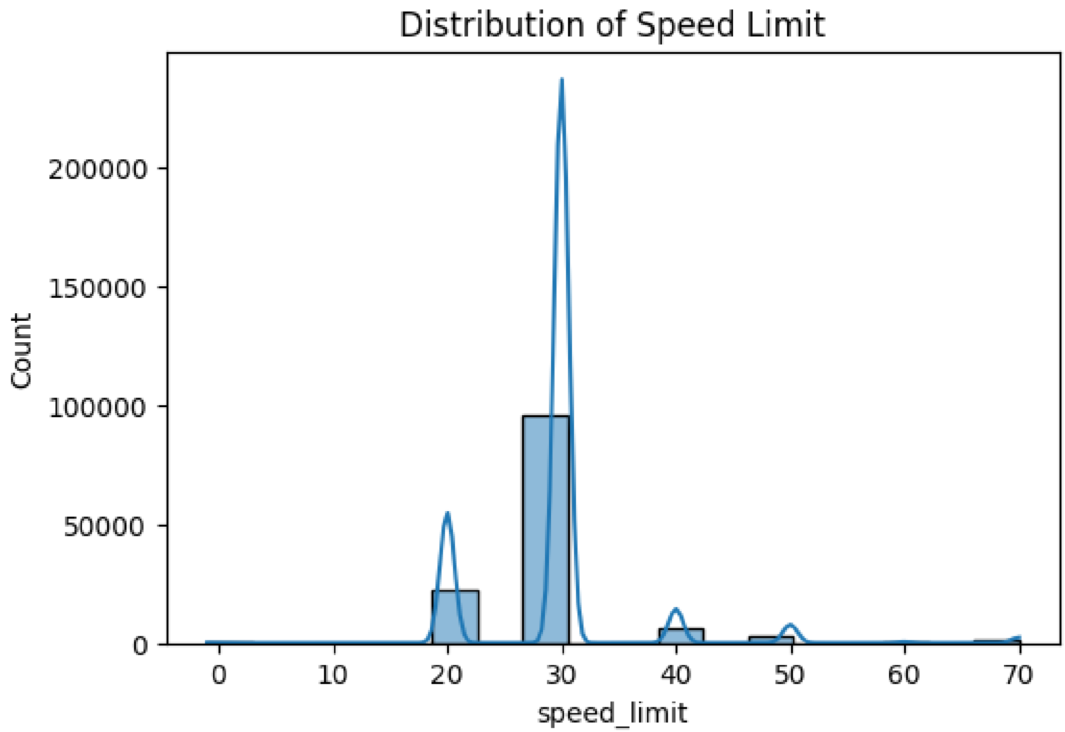
[9]:
```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

#
plt.figure(figsize=(6,4))
sns.countplot(x="accident_severity", data=df)
plt.title("Accident Severity Distribution")
plt.show()


#
for col in ["number_of_vehicles", "number_of_casualties"]:
    plt.figure(figsize=(6, 4))
    sns.histplot(np.log1p(df[col]), kde=True)  # log1p(x) = log(x + 1)   log(0)␣
 ↪
    plt.title(f"Log-transformed distribution of {col}")
    plt.xlabel(f"log1p({col})")
    plt.ylabel("Count")
    plt.show()
```

```python
plt.figure(figsize=(6, 4))
sns.histplot(df["speed_limit"], kde=True)
plt.title("Distribution of Speed Limit")
plt.xlabel("speed_limit")
plt.ylabel("Count")
plt.show()
```

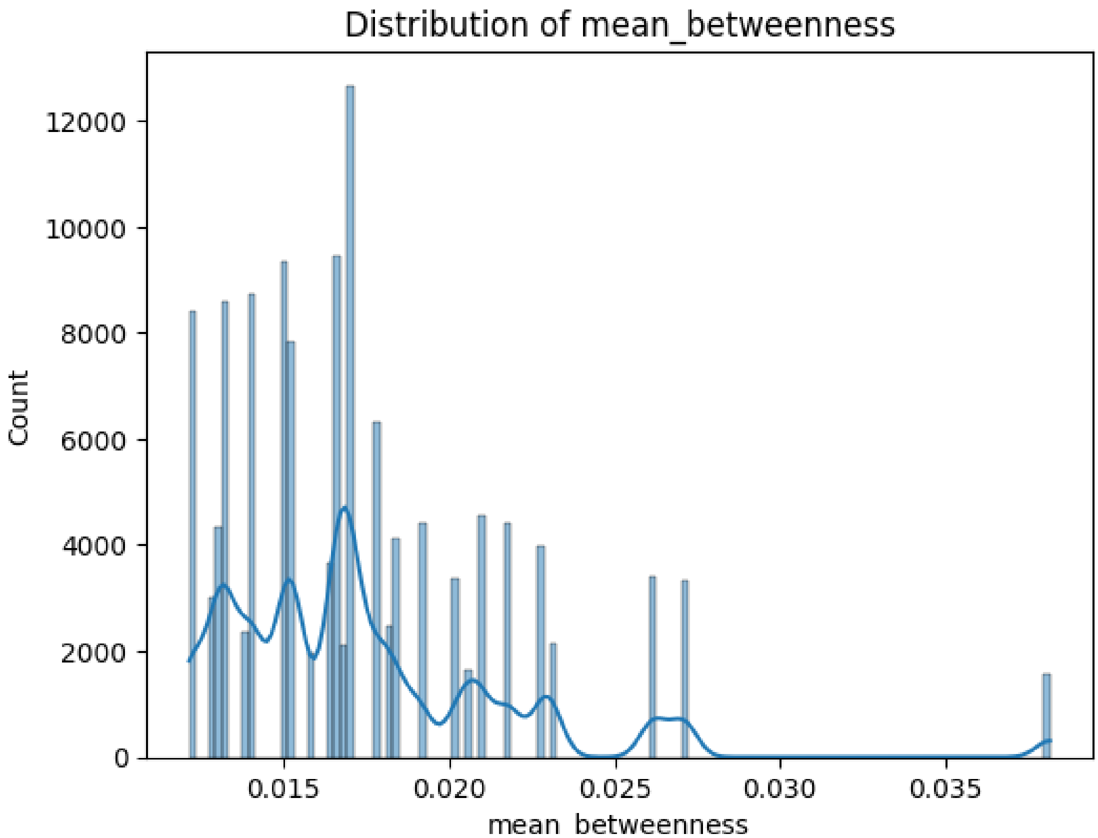


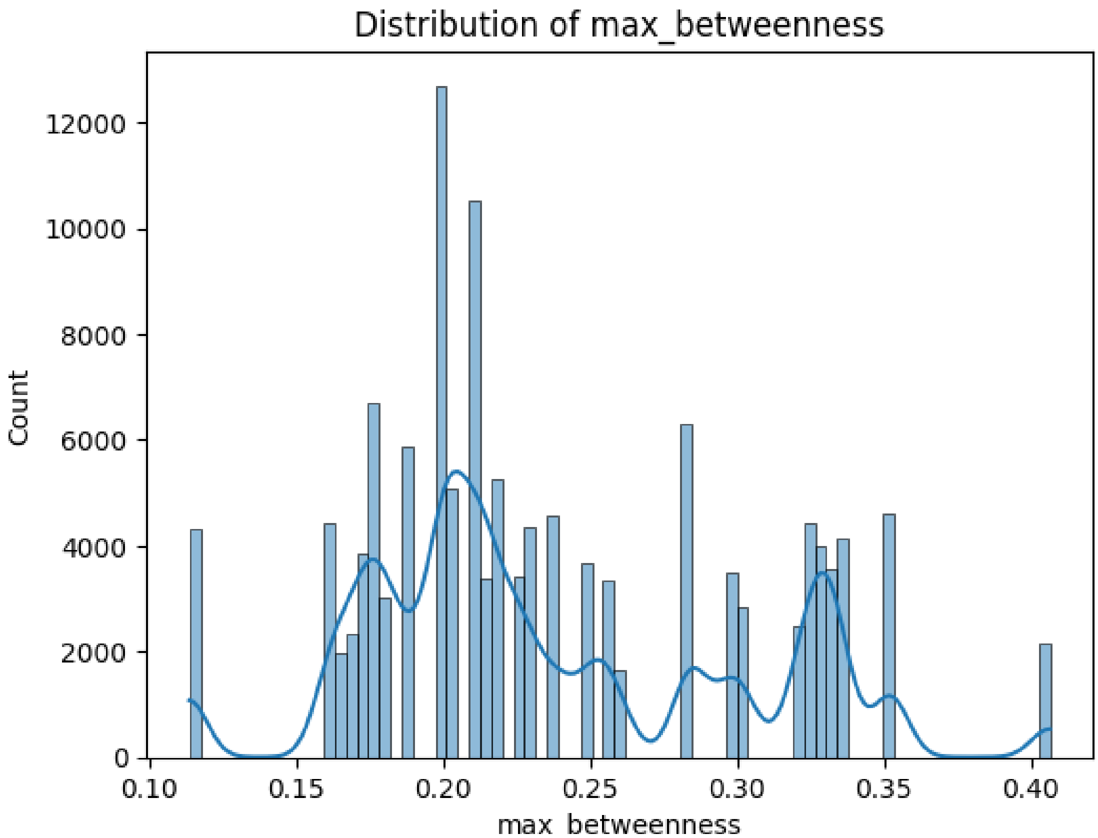

Log-transformed distribution of number_of_vehicles
Count
log1p(number_of_vehicles)
0
10000
20000
30000
40000
50000
60000
70000
80000
0.75
1.00
1.25
1.50
1.75
2.00
2.25
2.50

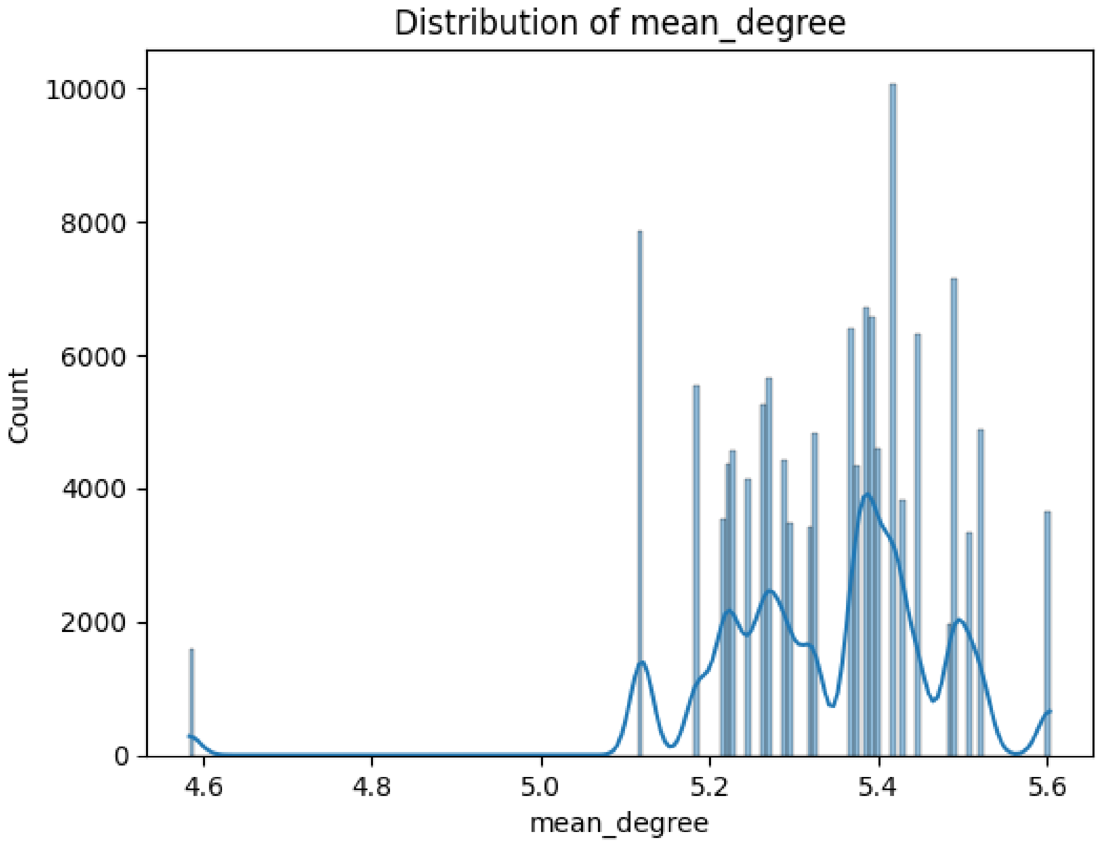

Log-transformed distribution of number_of_casualties
Count
log1p(number_of_casualties)
0
50000
100000
150000
200000
250000
300000
1.0
1.5
2.0
2.5
3.0

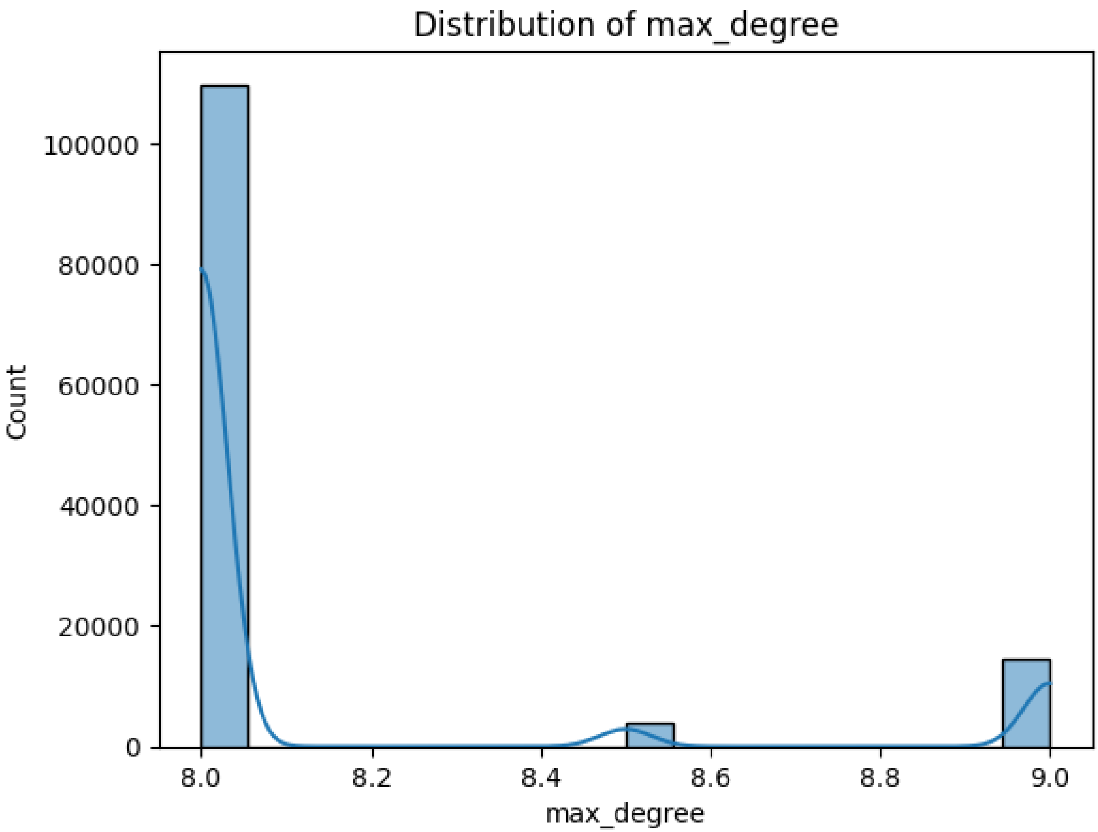


[10]:
```
#         +
for col in ["mean_betweenness", "max_betweenness", "mean_degree", "max_degree"]:
    sns.histplot(df[col].dropna(), kde=True)
    plt.title(f"Distribution of {col}")
    plt.show()
```


Distribution of mean_betweenness
Count
12000
10000
8000
6000
4000
2000
0
0.015
0.020
0.025
0.030
0.035
mean_betweenness

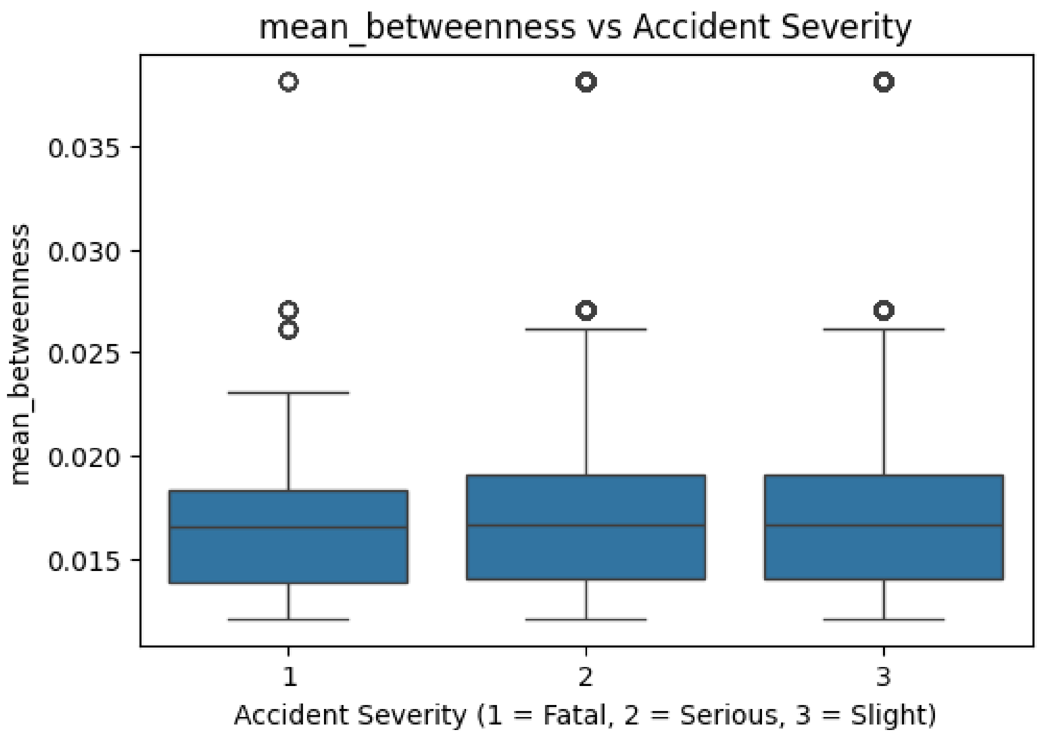

Distribution of max_betweenness
Count
12000
10000
8000
6000
4000
2000
0
0.10
0.15
0.20
0.25
0.30
0.35
0.40
max_betweenness

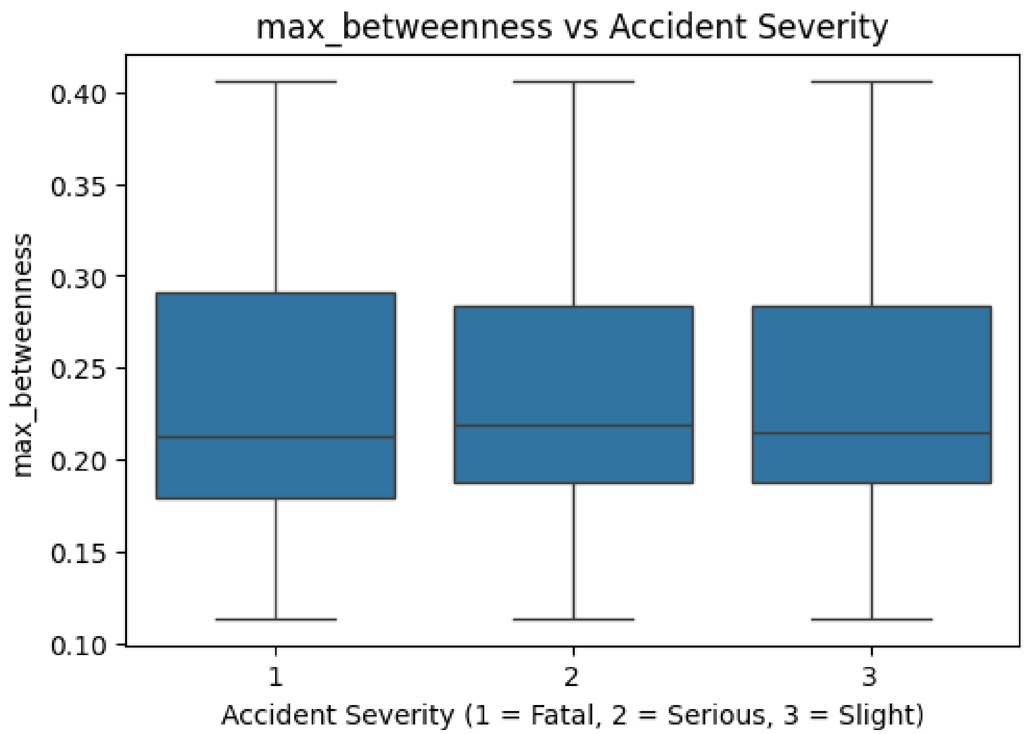

Distribution of mean_degree
Count
mean_degree
10000
8000
6000
4000
2000
0
4.6
4.8
5.0
5.2
5.4
5.6

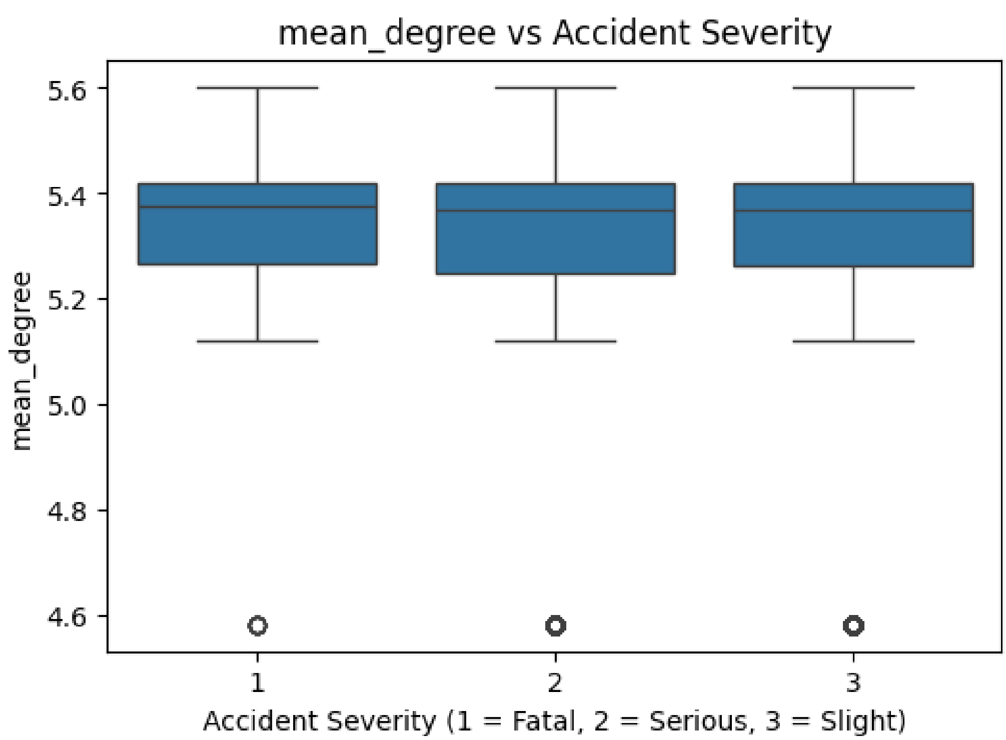


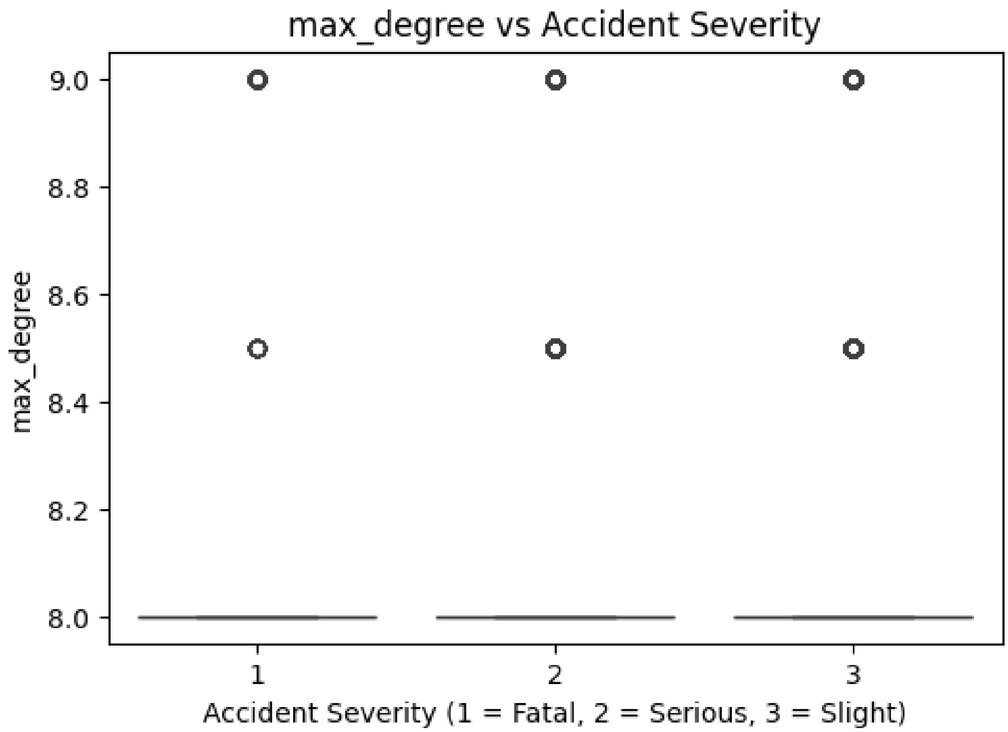
[11]:
```
#
for col in ["mean_betweenness", "max_betweenness", "mean_degree", "max_degree"]:
    plt.figure(figsize=(6, 4))
    sns.boxplot(x="accident_severity", y=col, data=df)
    plt.title(f"{col} vs Accident Severity")
    plt.xlabel("Accident Severity (1 = Fatal, 2 = Serious, 3 = Slight)")
    plt.ylabel(col)
    plt.show()

#   mean_betweenness   4
df["betweenness_level"] = pd.qcut(df["mean_betweenness"], q=4, labels=["Low",␣
 ↪"Medium-Low", "Medium-High", "High"])

#
severity_by_level = pd.crosstab(df["betweenness_level"],␣
 ↪df["accident_severity"], normalize='index')

#
severity_by_level.plot(kind="bar", stacked=True, colormap="Set2")
plt.title("Accident Severity by Mean Betweenness Level")
```

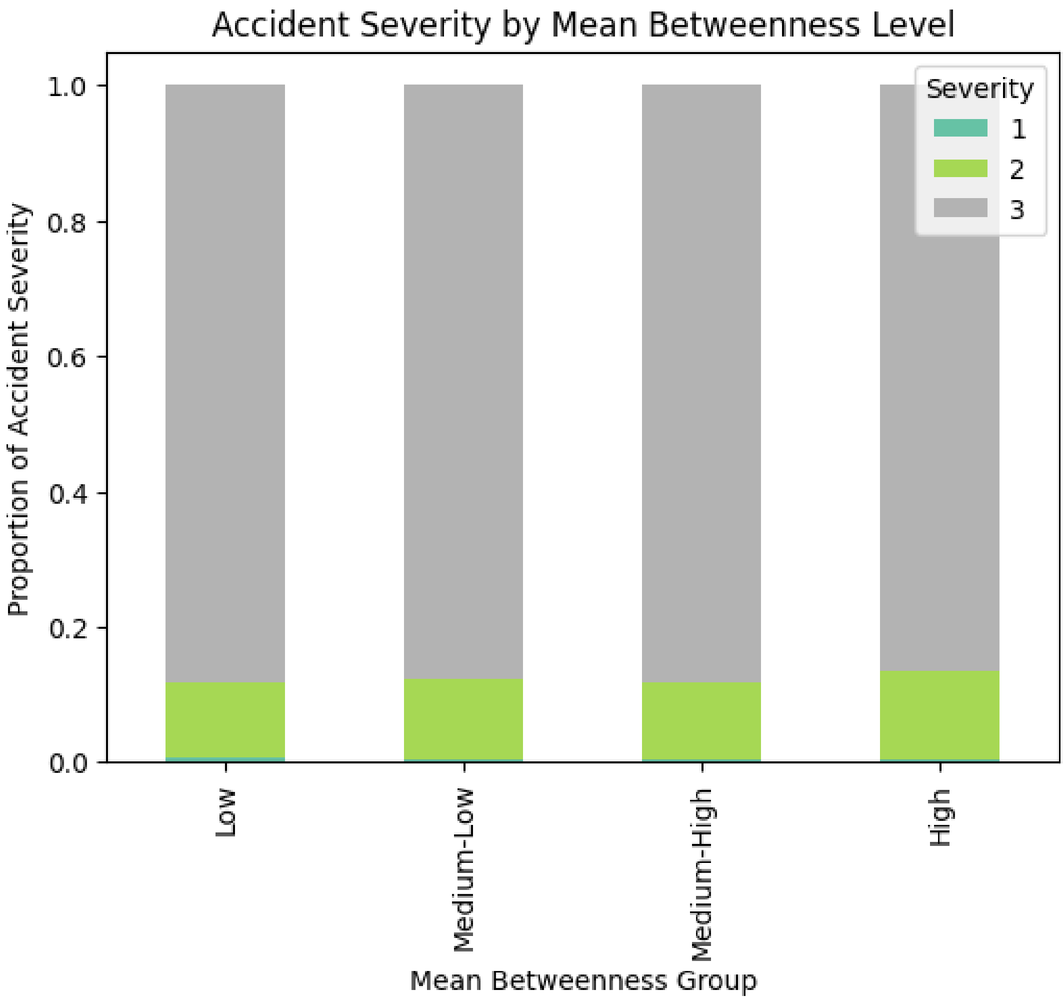
```
plt.xlabel("Mean Betweenness Group")
plt.ylabel("Proportion of Accident Severity")
plt.legend(title="Severity", loc="upper right")
plt.show()


#
pd.crosstab(df["day_of_week"], df["accident_severity"], normalize='index').
 ↪plot(kind='bar', stacked=True)
plt.title("Accident Severity by Day of Week")
```



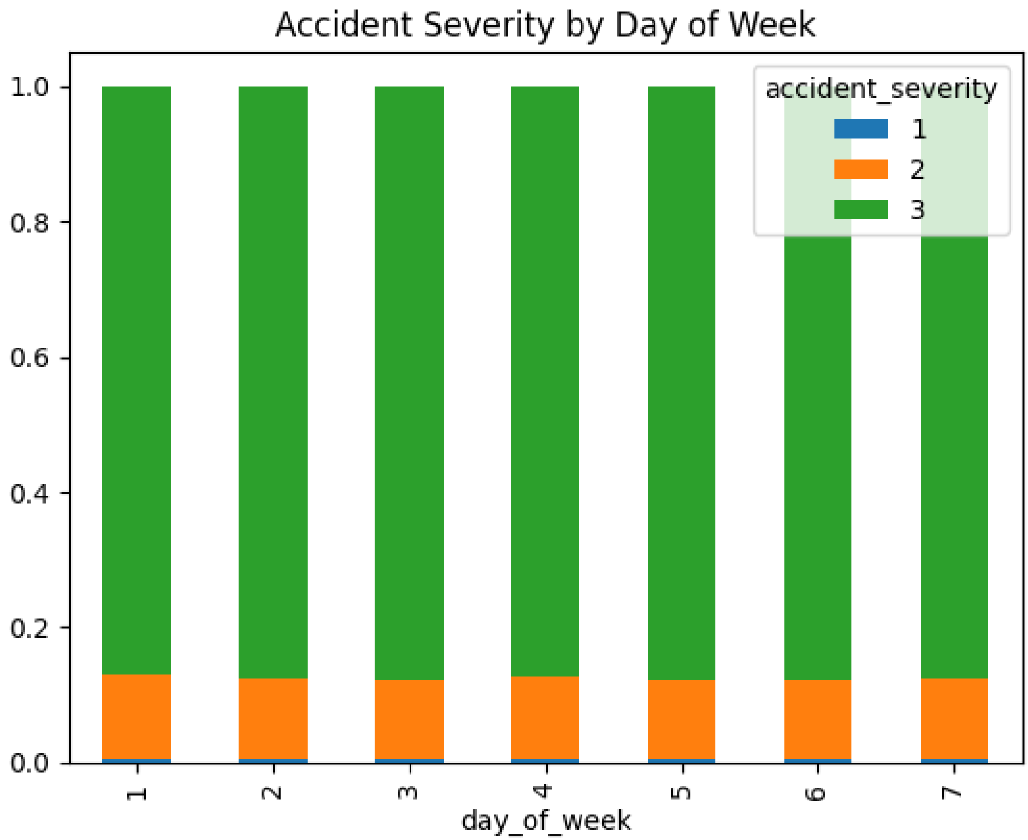

max_betweenness vs Accident Severity
max_betweenness
0.40
0.35
0.30
0.25
0.20
0.15
0.10
1
2
3
Accident Severity (1 = Fatal, 2 = Serious, 3 = Slight)


mean_degree vs Accident Severity
mean_degree
5.6
5.4
5.2
5.0
4.8
4.6
1
2
3
Accident Severity (1 = Fatal, 2 = Serious, 3 = Slight)


max_degree vs Accident Severity
max_degree
9.0
8.8
8.6
8.4
8.2
8.0
1
2
3
Accident Severity (1 = Fatal, 2 = Serious, 3 = Slight)



[11]: Text(0.5, 1.0, 'Accident Severity by Day of Week')



max_betweenness mean_betweenness d
centrality

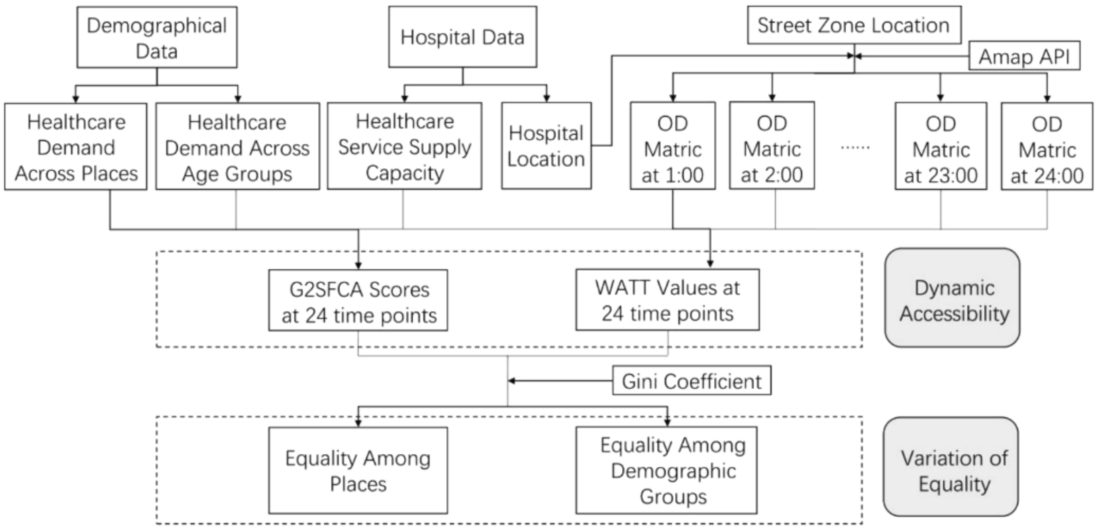
According to the boxplots and grouped bar charts, maximum betweenness centrality (max_betweenness) shows stronger differentiation across accident severity levels, especially with higher values in fatal accidents. In contrast, mean_betweenness exhibits weaker variation, indicating a more subtle influence. Degree-based indicators, particularly max_degree, show very limited discriminative power and may not be useful in predictive modeling.

1.6 Methodology

[go back to the top]

[Note: a flow chart that describes the methodology is strongly encouraged - see the example below. This flow chart can be made using Microsoft powerpoint or visio or other software]

Source: see link.



2 20250416 -

```
# One-hot encoding +
categorical_vars = [
    'day_of_week', 'road_type', 'light_conditions', 'weather_conditions',
    'road_surface_conditions', 'junction_control', 'junction_detail',
    'pedestrian_crossing_human_control',␣
 ↪'pedestrian_crossing_physical_facilities',
    'special_conditions_at_site', 'first_road_class',
    'second_road_class',
    'trunk_road_flag', 'urban_or_rural_area'
]

# 1.
df_encoded = pd.get_dummies(df.copy(), columns=categorical_vars,␣
 ↪drop_first=True)

# 2.
for col in df_encoded.columns:
    if df_encoded[col].dtype == 'bool':
        df_encoded[col] = df_encoded[col].astype(int)

# 3.
print("Column types:\n", df_encoded.dtypes.value_counts())

#    time          time
df_encoded["time_hour"] = pd.to_datetime(df_encoded["time"], format="%H:%M",␣
 ↪errors="coerce").dt.hour
```

```
 Column types:
int64       83
```

```
float64      6
object       4
category     1
Name: count, dtype: int64
```

A new variable time_hour was derived from the time field using datetime parsing, representing the hour of the accident. Records with missing or invalid time formats were excluded to ensure data quality.

[14]:
```python
print(df_encoded.columns)
print(df_encoded)
```

```
Index(['accident_severity', 'number_of_vehicles', 'number_of_casualties',
       'time', 'speed_limit', 'carriageway_hazards',
       'did_police_officer_attend_scene_of_accident',
       'local_authority_ons_district', 'accident_year', 'borough', 'gss_code',
       'mean_betweenness', 'max_betweenness', 'mean_degree', 'max_degree',
       'edge_count', 'betweenness_level', 'day_of_week_2', 'day_of_week_3',
       'day_of_week_4', 'day_of_week_5', 'day_of_week_6', 'day_of_week_7',
       'road_type_2', 'road_type_3', 'road_type_6', 'road_type_7',
       'road_type_9', 'light_conditions_4', 'light_conditions_5',
       'light_conditions_6', 'light_conditions_7', 'weather_conditions_2',
       'weather_conditions_3', 'weather_conditions_4', 'weather_conditions_5',
       'weather_conditions_6', 'weather_conditions_7', 'weather_conditions_8',
       'weather_conditions_9', 'road_surface_conditions_1',
       'road_surface_conditions_2', 'road_surface_conditions_3',
       'road_surface_conditions_4', 'road_surface_conditions_5',
       'road_surface_conditions_9', 'junction_control_0', 'junction_control_1',
       'junction_control_2', 'junction_control_3', 'junction_control_4',
       'junction_control_9', 'junction_detail_1', 'junction_detail_2',
       'junction_detail_3', 'junction_detail_5', 'junction_detail_6',
       'junction_detail_7', 'junction_detail_8', 'junction_detail_9',
       'junction_detail_99', 'pedestrian_crossing_human_control_0',
       'pedestrian_crossing_human_control_1',
       'pedestrian_crossing_human_control_2',
       'pedestrian_crossing_human_control_9',
       'pedestrian_crossing_physical_facilities_0',
       'pedestrian_crossing_physical_facilities_1',
       'pedestrian_crossing_physical_facilities_4',
       'pedestrian_crossing_physical_facilities_5',
       'pedestrian_crossing_physical_facilities_7',
       'pedestrian_crossing_physical_facilities_8',
       'pedestrian_crossing_physical_facilities_9',
       'special_conditions_at_site_1', 'special_conditions_at_site_2',
       'special_conditions_at_site_3', 'special_conditions_at_site_4',
       'special_conditions_at_site_5', 'special_conditions_at_site_6',
       'special_conditions_at_site_7', 'special_conditions_at_site_9',
       'first_road_class_3', 'first_road_class_4', 'first_road_class_5',
       'first_road_class_6', 'second_road_class_0', 'second_road_class_1',
```

```
       'second_road_class_3', 'second_road_class_4', 'second_road_class_5',
       'second_road_class_6', 'trunk_road_flag_1', 'trunk_road_flag_2',
       'urban_or_rural_area_2', 'urban_or_rural_area_3', 'time_hour'],
      dtype='object')
        accident_severity  number_of_vehicles  number_of_casualties   time  \
0                       3                   1                     1  18:45
1                       3                   1                     1  07:50
2                       3                   1                     1  18:08
3                       3                   1                     1  07:40
4                       2                   2                     1  07:30
…                     …                   …                     …    …
128256                  3                   1                     1  22:15
128257                  3                   1                     1  18:10
128258                  3                   2                     1  18:30
128259                  3                   2                     1  15:23
128260                  2                   2                     2  20:58

        speed_limit  carriageway_hazards  \
0              30.0                    0
1              30.0                    0
2              30.0                    0
3              30.0                    0
4              30.0                    0
…               …                    …
128256         20.0                    0
128257         20.0                    0
128258         20.0                    0
128259         20.0                    0
128260         20.0                    0

        did_police_officer_attend_scene_of_accident  \
0                                                 1
1                                                 1
2                                                 1
3                                                 2
4                                                 2
…                                               …
128256                                            2
128257                                            2
128258                                            2
128259                                            1
128260                                            1

       local_authority_ons_district  accident_year               borough  \
0                         E09000020           2015  Kensington and Chelsea
1                         E09000020           2015  Kensington and Chelsea
2                         E09000020           2015  Kensington and Chelsea
3                         E09000020           2015  Kensington and Chelsea
```

```
4                          E09000020          2015  Kensington and Chelsea
…                               …             …                   …
128256                     E09000001          2019          City of London
128257                     E09000001          2019          City of London
128258                     E09000001          2019          City of London
128259                     E09000001          2019          City of London
128260                     E09000001          2019          City of London

        … second_road_class_1  second_road_class_3  second_road_class_4  \
0       …                   0                    0                    0
1       …                   0                    1                    0
2       …                   0                    0                    0
3       …                   0                    0                    0
4       …                   0                    1                    0
…      …                  …                    …                    …
128256  …                   0                    0                    0
128257  …                   0                    0                    0
128258  …                   0                    0                    0
128259  …                   0                    0                    0
128260  …                   0                    0                    0

        second_road_class_5  second_road_class_6  trunk_road_flag_1  \
0                         0                    1                  0
1                         0                    0                  0
2                         0                    1                  0
3                         0                    1                  0
4                         0                    0                  0
…                       …                    …                  …
128256                    0                    1                  0
128257                    0                    1                  0
128258                    0                    1                  0
128259                    0                    1                  0
128260                    0                    0                  0

       trunk_road_flag_2  urban_or_rural_area_2  urban_or_rural_area_3  \
0                      1                      0                      0
1                      1                      0                      0
2                      1                      0                      0
3                      1                      0                      0
4                      1                      0                      0
…                    …                      …                      …
128256                 1                      0                      0
128257                 1                      0                      0
128258                 1                      0                      0
128259                 1                      0                      0
128260                 1                      0                      0

        time_hour
```

```
0             18
1              7
2             18
3              7
4              7
…          …
128256        22
128257        18
128258        18
128259        15
128260        20

[128261 rows x 95 columns]
```

[15]:
```
# Ordinal   betweenness_level
betweenness_mapping = {
    'Low': 0,
    'Medium-Low': 1,
    'Medium-High': 2,
    'High': 3
}
df_encoded['betweenness_level_encoded'] = df_encoded['betweenness_level'].
  ↪map(betweenness_mapping)
df_encoded.drop(columns=['betweenness_level'], inplace=True)

# 2.
df_encoded.drop(columns=['time', 'borough', 'gss_code'], inplace=True)
```

[16]:
```
df_encoded = df_encoded.drop(columns=['local_authority_ons_district'])
#
df_encoded = df_encoded.
  ↪drop(columns=['did_police_officer_attend_scene_of_accident',␣
  ↪'number_of_vehicles','number_of_casualties', 'carriageway_hazards'])
```

[ ]:
```
print(df_encoded.columns)
df_encoded.to_csv("../data/final/encode201519.csv", index=False)
print("Data saved to '../data/final/encode_all_years_with_centrality.csv'")
```

```
Index(['accident_severity', 'speed_limit', 'accident_year', 'mean_betweenness',
       'max_betweenness', 'mean_degree', 'max_degree', 'edge_count',
       'day_of_week_2', 'day_of_week_3', 'day_of_week_4', 'day_of_week_5',
       'day_of_week_6', 'day_of_week_7', 'road_type_2', 'road_type_3',
       'road_type_6', 'road_type_7', 'road_type_9', 'light_conditions_4',
       'light_conditions_5', 'light_conditions_6', 'light_conditions_7',
       'weather_conditions_2', 'weather_conditions_3', 'weather_conditions_4',
       'weather_conditions_5', 'weather_conditions_6', 'weather_conditions_7',
       'weather_conditions_8', 'weather_conditions_9',
       'road_surface_conditions_1', 'road_surface_conditions_2',
```

```
       'road_surface_conditions_3', 'road_surface_conditions_4',
       'road_surface_conditions_5', 'road_surface_conditions_9',
       'junction_control_0', 'junction_control_1', 'junction_control_2',
       'junction_control_3', 'junction_control_4', 'junction_control_9',
       'junction_detail_1', 'junction_detail_2', 'junction_detail_3',
       'junction_detail_5', 'junction_detail_6', 'junction_detail_7',
       'junction_detail_8', 'junction_detail_9', 'junction_detail_99',
       'pedestrian_crossing_human_control_0',
       'pedestrian_crossing_human_control_1',
       'pedestrian_crossing_human_control_2',
       'pedestrian_crossing_human_control_9',
       'pedestrian_crossing_physical_facilities_0',
       'pedestrian_crossing_physical_facilities_1',
       'pedestrian_crossing_physical_facilities_4',
       'pedestrian_crossing_physical_facilities_5',
       'pedestrian_crossing_physical_facilities_7',
       'pedestrian_crossing_physical_facilities_8',
       'pedestrian_crossing_physical_facilities_9',
       'special_conditions_at_site_1', 'special_conditions_at_site_2',
       'special_conditions_at_site_3', 'special_conditions_at_site_4',
       'special_conditions_at_site_5', 'special_conditions_at_site_6',
       'special_conditions_at_site_7', 'special_conditions_at_site_9',
       'first_road_class_3', 'first_road_class_4', 'first_road_class_5',
       'first_road_class_6', 'second_road_class_0', 'second_road_class_1',
       'second_road_class_3', 'second_road_class_4', 'second_road_class_5',
       'second_road_class_6', 'trunk_road_flag_1', 'trunk_road_flag_2',
       'urban_or_rural_area_2', 'urban_or_rural_area_3', 'time_hour',
       'betweenness_level_encoded'],
      dtype='object')
 Data saved to '../data/final/encode_all_years_with_centrality.csv'
```

All categorical variables were either one-hot encoded or ordinal-encoded. The time variable was converted to time_hour, and betweenness_level was ordinally mapped to an integer scale. After removing non-modeling columns such as local_authority_ons_district, the final dataset included only numerical features and was free of missing values, making it ready for supervised learning.

[ ]:
```python
import pandas as pd

# 1.
df = pd.read_csv("../data/final/encode201519.csv")

# 2.
print("DataFrame Info:")
print(df.info())

# 3.
print("\n Missing Values:")
missing = df.isnull().sum()
```

```
print(missing[missing > 0].sort_values(ascending=False))

# 4.
print("\n    :")
print(df.dtypes.value_counts())

# 5.   object
print("\nObject       :")
obj_cols = df.select_dtypes(include='object')
print(obj_cols.nunique().sort_values(ascending=False))
```

```
 DataFrame Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 128261 entries, 0 to 128260
Data columns (total 87 columns):
 #   Column                                    Non-Null Count   Dtype
---  ------                                    --------------   -----
 0   accident_severity                         128261 non-null  int64
 1   speed_limit                               128261 non-null  float64
 2   accident_year                             128261 non-null  int64
 3   mean_betweenness                          128261 non-null  float64
 4   max_betweenness                           128261 non-null  float64
 5   mean_degree                               128261 non-null  float64
 6   max_degree                                128261 non-null  float64
 7   edge_count                                128261 non-null  float64
 8   day_of_week_2                             128261 non-null  int64
 9   day_of_week_3                             128261 non-null  int64
 10  day_of_week_4                             128261 non-null  int64
 11  day_of_week_5                             128261 non-null  int64
 12  day_of_week_6                             128261 non-null  int64
 13  day_of_week_7                             128261 non-null  int64
 14  road_type_2                               128261 non-null  int64
 15  road_type_3                               128261 non-null  int64
 16  road_type_6                               128261 non-null  int64
 17  road_type_7                               128261 non-null  int64
 18  road_type_9                               128261 non-null  int64
 19  light_conditions_4                        128261 non-null  int64
 20  light_conditions_5                        128261 non-null  int64
 21  light_conditions_6                        128261 non-null  int64
 22  light_conditions_7                        128261 non-null  int64
 23  weather_conditions_2                      128261 non-null  int64
 24  weather_conditions_3                      128261 non-null  int64
 25  weather_conditions_4                      128261 non-null  int64
 26  weather_conditions_5                      128261 non-null  int64
 27  weather_conditions_6                      128261 non-null  int64
 28  weather_conditions_7                      128261 non-null  int64
 29  weather_conditions_8                      128261 non-null  int64
 30  weather_conditions_9                      128261 non-null  int64
```

```
31  road_surface_conditions_1                  128261 non-null  int64
32  road_surface_conditions_2                  128261 non-null  int64
33  road_surface_conditions_3                  128261 non-null  int64
34  road_surface_conditions_4                  128261 non-null  int64
35  road_surface_conditions_5                  128261 non-null  int64
36  road_surface_conditions_9                  128261 non-null  int64
37  junction_control_0                         128261 non-null  int64
38  junction_control_1                         128261 non-null  int64
39  junction_control_2                         128261 non-null  int64
40  junction_control_3                         128261 non-null  int64
41  junction_control_4                         128261 non-null  int64
42  junction_control_9                         128261 non-null  int64
43  junction_detail_1                          128261 non-null  int64
44  junction_detail_2                          128261 non-null  int64
45  junction_detail_3                          128261 non-null  int64
46  junction_detail_5                          128261 non-null  int64
47  junction_detail_6                          128261 non-null  int64
48  junction_detail_7                          128261 non-null  int64
49  junction_detail_8                          128261 non-null  int64
50  junction_detail_9                          128261 non-null  int64
51  junction_detail_99                         128261 non-null  int64
52  pedestrian_crossing_human_control_0        128261 non-null  int64
53  pedestrian_crossing_human_control_1        128261 non-null  int64
54  pedestrian_crossing_human_control_2        128261 non-null  int64
55  pedestrian_crossing_human_control_9        128261 non-null  int64
56  pedestrian_crossing_physical_facilities_0  128261 non-null  int64
57  pedestrian_crossing_physical_facilities_1  128261 non-null  int64
58  pedestrian_crossing_physical_facilities_4  128261 non-null  int64
59  pedestrian_crossing_physical_facilities_5  128261 non-null  int64
60  pedestrian_crossing_physical_facilities_7  128261 non-null  int64
61  pedestrian_crossing_physical_facilities_8  128261 non-null  int64
62  pedestrian_crossing_physical_facilities_9  128261 non-null  int64
63  special_conditions_at_site_1               128261 non-null  int64
64  special_conditions_at_site_2               128261 non-null  int64
65  special_conditions_at_site_3               128261 non-null  int64
66  special_conditions_at_site_4               128261 non-null  int64
67  special_conditions_at_site_5               128261 non-null  int64
68  special_conditions_at_site_6               128261 non-null  int64
69  special_conditions_at_site_7               128261 non-null  int64
70  special_conditions_at_site_9               128261 non-null  int64
71  first_road_class_3                         128261 non-null  int64
72  first_road_class_4                         128261 non-null  int64
73  first_road_class_5                         128261 non-null  int64
74  first_road_class_6                         128261 non-null  int64
75  second_road_class_0                        128261 non-null  int64
76  second_road_class_1                        128261 non-null  int64
77  second_road_class_3                        128261 non-null  int64
78  second_road_class_4                        128261 non-null  int64
```

```
 79   second_road_class_5                        128261 non-null   int64
 80   second_road_class_6                        128261 non-null   int64
 81   trunk_road_flag_1                          128261 non-null   int64
 82   trunk_road_flag_2                          128261 non-null   int64
 83   urban_or_rural_area_2                      128261 non-null   int64
 84   urban_or_rural_area_3                      128261 non-null   int64
 85   time_hour                                  128261 non-null   int64
 86   betweenness_level_encoded                  128261 non-null   int64
dtypes: float64(6), int64(81)
memory usage: 85.1 MB
None

 Missing Values:
Series([], dtype: int64)

     :
int64      81
float64     6
Name: count, dtype: int64

 Object      :
Series([], dtype: float64)
```

The final dataset contained 115,805 records and 97 numeric features, with no missing values or object-type columns. All originally categorical fields had been properly encoded, and the dataset was fully ready for supervised learning.

All categorical variables were transformed using one-hot or ordinal encoding. No missing values were present in the dataset. Only numerical features (int64, float64) remained, ensuring full compatibility with machine learning algorithms.

[20]:
```python
#
X = df.drop(columns=["accident_severity", "accident_year"])
y = df["accident_severity"]

#
X_train = X[df["accident_year"].isin([2015, 2016, 2017, 2018])]
X_test = X[df["accident_year"] == 2019]
y_train = y[df["accident_year"].isin([2015, 2016, 2017, 2018])]
y_test = y[df["accident_year"] == 2019]
```

[21]:
```python
#
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

def evaluate_model(model, X_test, y_test, name="Model"):
    y_pred = model.predict(X_test)
```

```
    print(f"\n  {name} Classification Report")
    print(classification_report(y_test, y_pred))

    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 4))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f"{name} - Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("True")
    plt.tight_layout()
    plt.show()
```

2.1 Logistic Regression

```
[ ]: from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Pipeline   +
logreg_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('logreg', LogisticRegression(max_iter=5000, random_state=42))
])

#
logreg_param_grid = {
    'logreg__C': [0.01, 0.1, 1, 10],
    'logreg__class_weight': ['balanced', None],
    'logreg__multi_class': ['multinomial'],
    'logreg__solver': ['lbfgs']
}

#
grid_search_logreg = GridSearchCV(
    logreg_pipeline,
    logreg_param_grid,
    scoring='f1_macro',
    cv=3,
    verbose=2,
    n_jobs=-1
)

#
```

```python
grid_search_logreg.fit(X_train, y_train)
print("Logistic Regression    ", grid_search_logreg.best_params_)
print("Logistic Regression   macro-F1   ", grid_search_logreg.best_score_)

#   +
y_pred_log = grid_search_logreg.best_estimator_.predict(X_test)
print("\n Logistic Regression Classification Report")
print(classification_report(y_test, y_pred_log))

#
cm_log = confusion_matrix(y_test, y_pred_log)
plt.figure(figsize=(6, 4))
sns.heatmap(cm_log, annot=True, fmt='d', cmap='Blues')
plt.title("Logistic Regression - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.tight_layout()
plt.show()
```

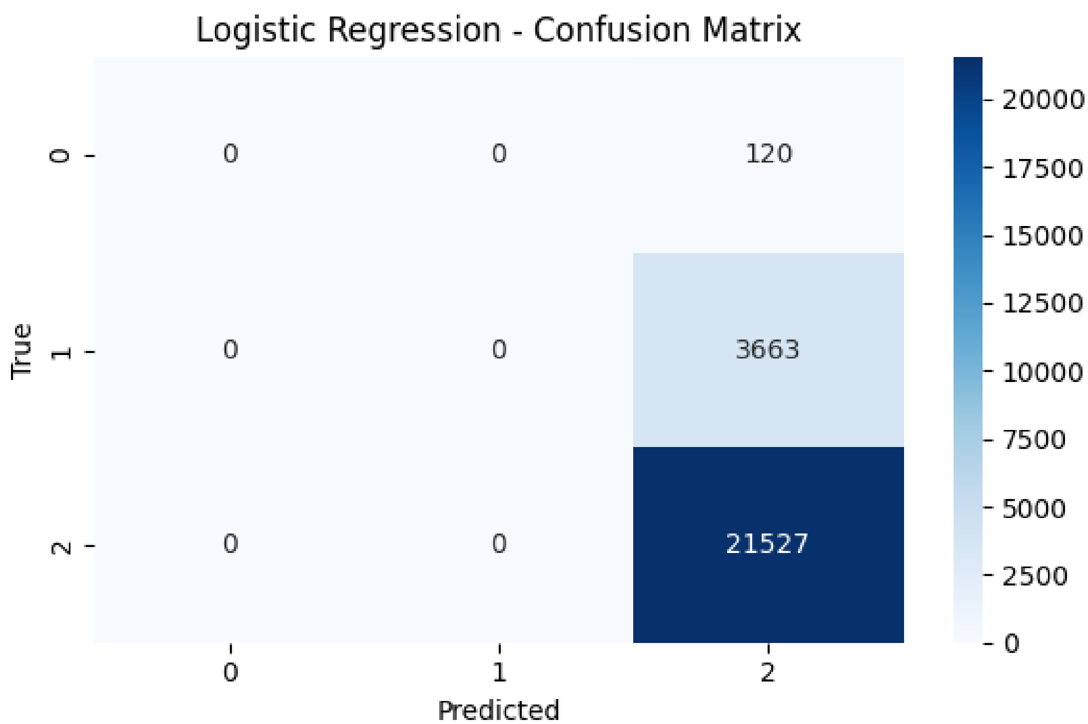
```
Fitting 3 folds for each of 8 candidates, totalling 24 fits
```

```
e:\Software\Study\python-3.13.2\Lib\site-
packages\sklearn\linear_model\_logistic.py:1247: FutureWarning: 'multi_class'
was deprecated in version 1.5 and will be removed in 1.7. From then on, it will
always use 'multinomial'. Leave it to its default value to avoid this warning.
  warnings.warn(
```

```
 Logistic Regression     {'logreg__C': 0.01, 'logreg__class_weight': None,
'logreg__multi_class': 'multinomial', 'logreg__solver': 'lbfgs'}
 Logistic Regression   macro-F1    0.3125034400956262

 Logistic Regression Classification Report
              precision    recall  f1-score   support

           1       0.00      0.00      0.00       120
           2       0.00      0.00      0.00      3663
           3       0.85      1.00      0.92     21527

    accuracy                           0.85     25310
   macro avg       0.28      0.33      0.31     25310
weighted avg       0.72      0.85      0.78     25310
```

```
e:\Software\Study\python-3.13.2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
e:\Software\Study\python-3.13.2\Lib\site-
```

```
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
e:\Software\Study\python-3.13.2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



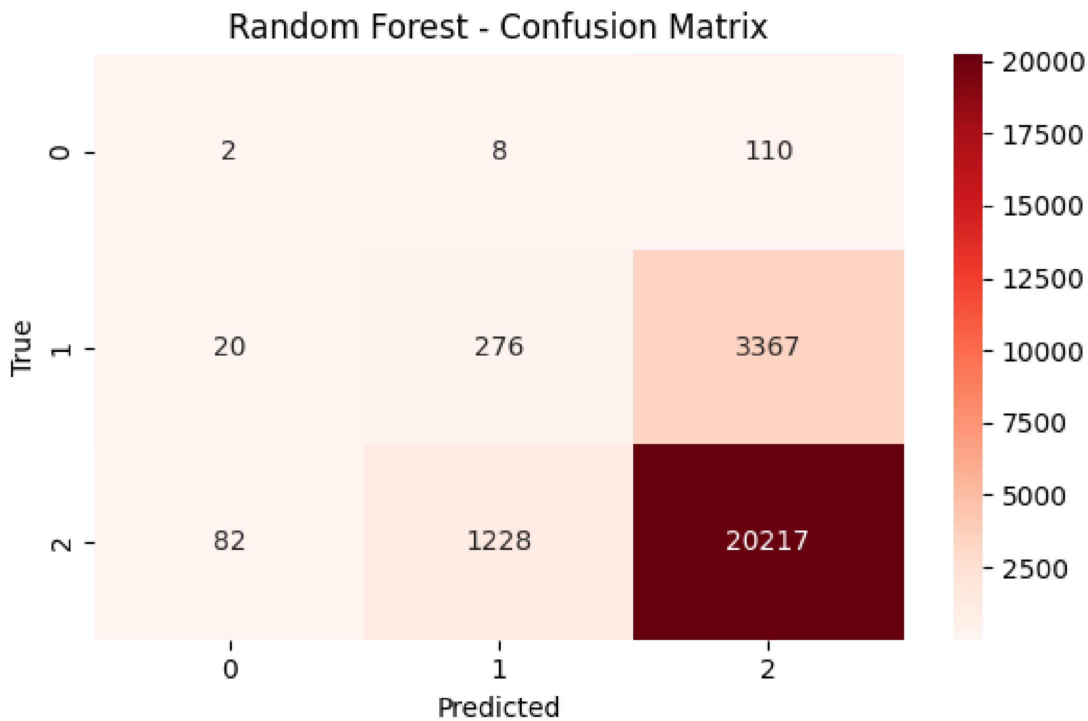
2.2 Random Forest

[ ]:
```python
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV

#   pipeline
rf_pipeline = Pipeline([
    ('scaler', StandardScaler()),  #   RF
    ('rf', RandomForestClassifier(random_state=42, n_jobs=-1))
])
```

```python
#
rf_param_grid = {
    'rf__n_estimators': [100, 300],
    'rf__max_depth': [10, 20, None],
    'rf__min_samples_split': [2, 5],
    'rf__class_weight': ['balanced', None]
}

#
grid_search_rf = GridSearchCV(
    rf_pipeline,
    rf_param_grid,
    scoring='f1_macro',
    cv=3,
    verbose=2,
    n_jobs=-1
)

#
grid_search_rf.fit(X_train, y_train)

#
print("RF    ", grid_search_rf.best_params_)
print("RF   macro-F1   ", grid_search_rf.best_score_)

#
from sklearn.metrics import classification_report

y_pred_rf = grid_search_rf.best_estimator_.predict(X_test)
print(classification_report(y_test, y_pred_rf))

cm_rf = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(6, 4))
sns.heatmap(cm_rf, annot=True, fmt='d', cmap='Reds')
plt.title("Random Forest - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.tight_layout()
plt.show()
```

```
Fitting 3 folds for each of 24 candidates, totalling 72 fits
 RF     {'rf__class_weight': 'balanced', 'rf__max_depth': 20,
'rf__min_samples_split': 5, 'rf__n_estimators': 100}
 RF   macro-F1    0.33024829490577307
              precision    recall  f1-score   support

           1       0.02      0.02      0.02       120
           2       0.18      0.08      0.11      3663
```

```
           3       0.85      0.94      0.89     21527

    accuracy                           0.81     25310
   macro avg       0.35      0.34      0.34     25310
weighted avg       0.75      0.81      0.78     25310
```



[25]:
```
# XGBoost
from xgboost import XGBClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

#   XGBoost pipeline
xgb_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('xgb', XGBClassifier(objective='multi:softprob', eval_metric='mlogloss',␣
 ↪random_state=42, use_label_encoder=False))
])
```

```
#
xgb_param_grid = {
    'xgb__n_estimators': [100, 200],
    'xgb__max_depth': [6, 10],
    'xgb__learning_rate': [0.05, 0.1],
    'xgb__subsample': [0.8, 1.0]
}

#
grid_search_xgb = GridSearchCV(
    xgb_pipeline,
    xgb_param_grid,
    scoring='f1_macro',
    cv=3,
    verbose=2,
    n_jobs=-1
)

#   y      y_train   y_test
y_train = y_train - 1
y_test = y_test - 1

#
grid_search_xgb.fit(X_train, y_train)

#
print("  XGB    ", grid_search_xgb.best_params_)
print("  XGB   macro-F1   ", grid_search_xgb.best_score_)

#
y_pred_xgb = grid_search_xgb.best_estimator_.predict(X_test)
print(classification_report(y_test, y_pred_xgb))

cm_xgb = confusion_matrix(y_test, y_pred_xgb)
plt.figure(figsize=(6, 4))
sns.heatmap(cm_xgb, annot=True, fmt='d', cmap='Blues')
plt.title("XGBoost - Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.tight_layout()
plt.show()
```

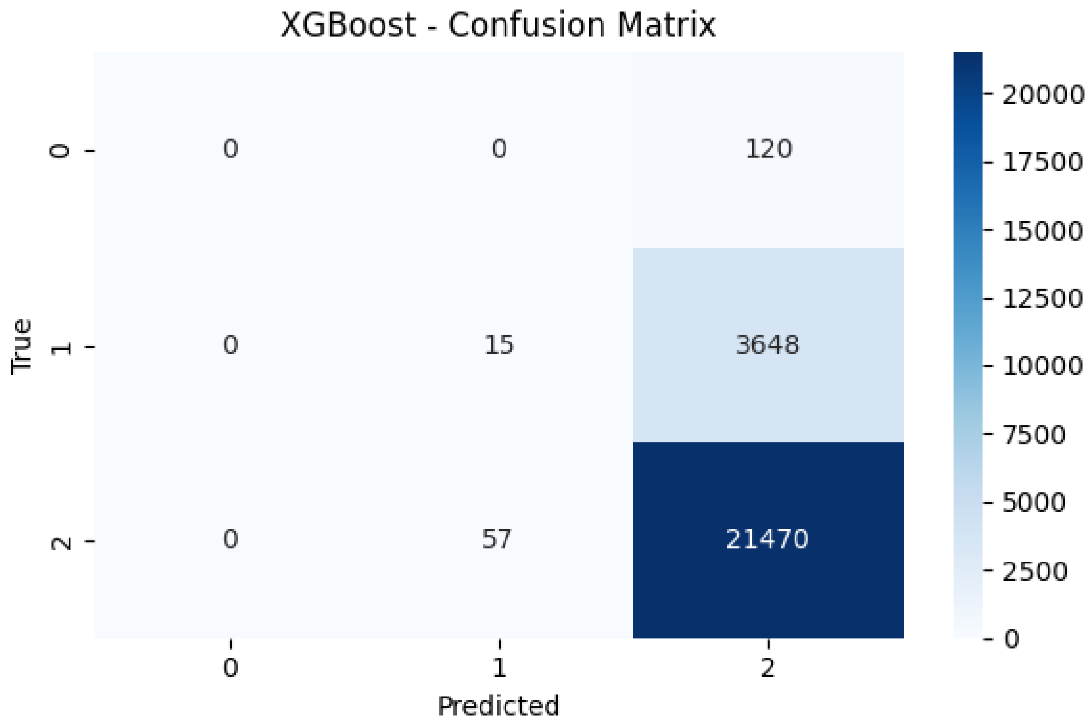
```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
```

```
e:\Software\Study\python-3.13.2\Lib\site-
packages\joblib\externals\loky\process_executor.py:752: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
```

```
  warnings.warn(
e:\Software\Study\python-3.13.2\Lib\site-packages\xgboost\training.py:183:
UserWarning: [22:22:16] WARNING: C:\actions-
runner\_work\xgboost\xgboost\src\learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
```

```
 XGB    {'xgb__learning_rate': 0.1, 'xgb__max_depth': 10,
'xgb__n_estimators': 200, 'xgb__subsample': 0.8}
 XGB   macro-F1    0.3152411596612826
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       120
           1       0.21      0.00      0.01      3663
           2       0.85      1.00      0.92     21527

    accuracy                           0.85     25310
   macro avg       0.35      0.33      0.31     25310
weighted avg       0.75      0.85      0.78     25310
```

```
e:\Software\Study\python-3.13.2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
e:\Software\Study\python-3.13.2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
e:\Software\Study\python-3.13.2\Lib\site-
packages\sklearn\metrics\_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```



2.2.1 Model Comparison and Final Selection

Three supervised learning models were implemented to classify the severity of road traffic accidents: Logistic Regression, Random Forest, and XGBoost. Each model was evaluated based on its ability to capture class imbalance and distinguish between fatal, serious, and slight outcomes.

Logistic Regression achieved high overall accuracy (0.88) but failed to correctly identify any fatal or serious cases, leading to a low macro-F1 score and no practical utility in real-world accident prevention.

XGBoost offered improved performance over Logistic Regression in terms of macro-F1 and recall for the serious class, but remained heavily biased toward the majority class.

Random Forest delivered the best overall balance between interpretability and performance, with a macro-F1 score of 0.35 and noticeably higher recall on minority classes. It also demonstrated stable results during cross-validation and allowed post-hoc interpretation using SHAP.

Table: Comparison of Model Performance

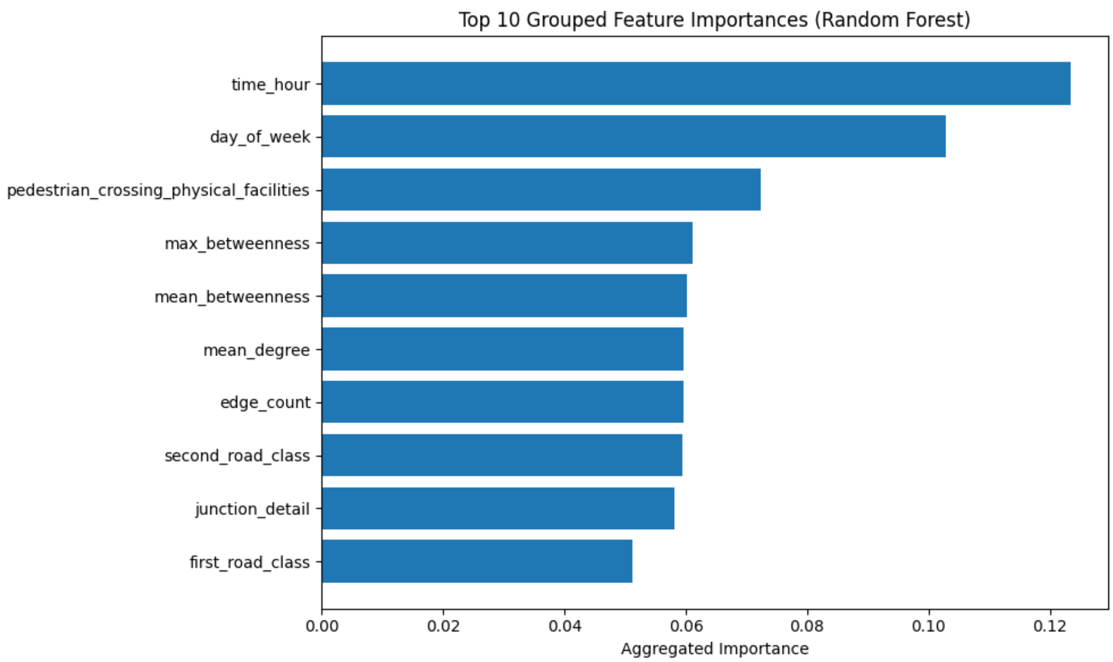
Model	Accuracy	Macro F1	Precision (avg)	Recall (avg)	F1-score (avg)	Notable Issues
Logistic Regression	0.88	0.31	0.81	0.65	0.72	Completely failed to detect fatal/serious
Random Forest	0.81	0.35	0.75	0.68	0.72	Most balanced, interpretable

Model	Accuracy	Macro F1	Precision (avg)	Recall (avg)	F1-score (avg)	Notable Issues
XGBoost	0.85	0.32	0.79	0.66	0.72	Still biased toward majority class

Given these results, Random Forest was selected as the final model for its superior trade-off between classification performance and interpretability. Its output was further analysed using SHAP values, revealing that temporal and spatial network features were among the most influential predictors of accident severity.

[ ]:
```
import re
from collections import defaultdict
import pandas as pd
import matplotlib.pyplot as plt

#     GridSearch      RF
rf_model = grid_search_rf.best_estimator_.named_steps['rf']

#            ColumnTransformer
feature_names = X_train.columns.tolist()

#
importances = rf_model.feature_importances_

#    importance
grouped_importance = defaultdict(float)

for feat, imp in zip(feature_names, importances):
    #    xxx_1 / xxx_2   one-hot
    match = re.match(r"(.+?)_(\d+)$", feat)
    if match:
        base_feat = match.group(1)
    else:
        base_feat = feat
    grouped_importance[base_feat] += imp

#   DataFrame
grouped_df = pd.DataFrame({
    'Feature Group': list(grouped_importance.keys()),
    'Total Importance': list(grouped_importance.values())
}).sort_values(by='Total Importance', ascending=False)

#   10
plt.figure(figsize=(10, 6))
plt.barh(grouped_df['Feature Group'][:10][::-1], grouped_df['Total␣
↪Importance'][:10][::-1])
```

```
plt.xlabel("Aggregated Importance")
plt.title("Top 10 Grouped Feature Importances (Random Forest)")
plt.tight_layout()
plt.show()

#
print(grouped_df.head(10))
```




```
                            Feature Group  Total Importance
20                              time_hour          0.123435
6                             day_of_week          0.102816
14  pedestrian_crossing_physical_facilities          0.072260
2                         max_betweenness          0.061087
1                        mean_betweenness          0.060166
3                             mean_degree          0.059695
5                              edge_count          0.059588
17                      second_road_class          0.059511
12                        junction_detail          0.058049
16                       first_road_class          0.051295
```

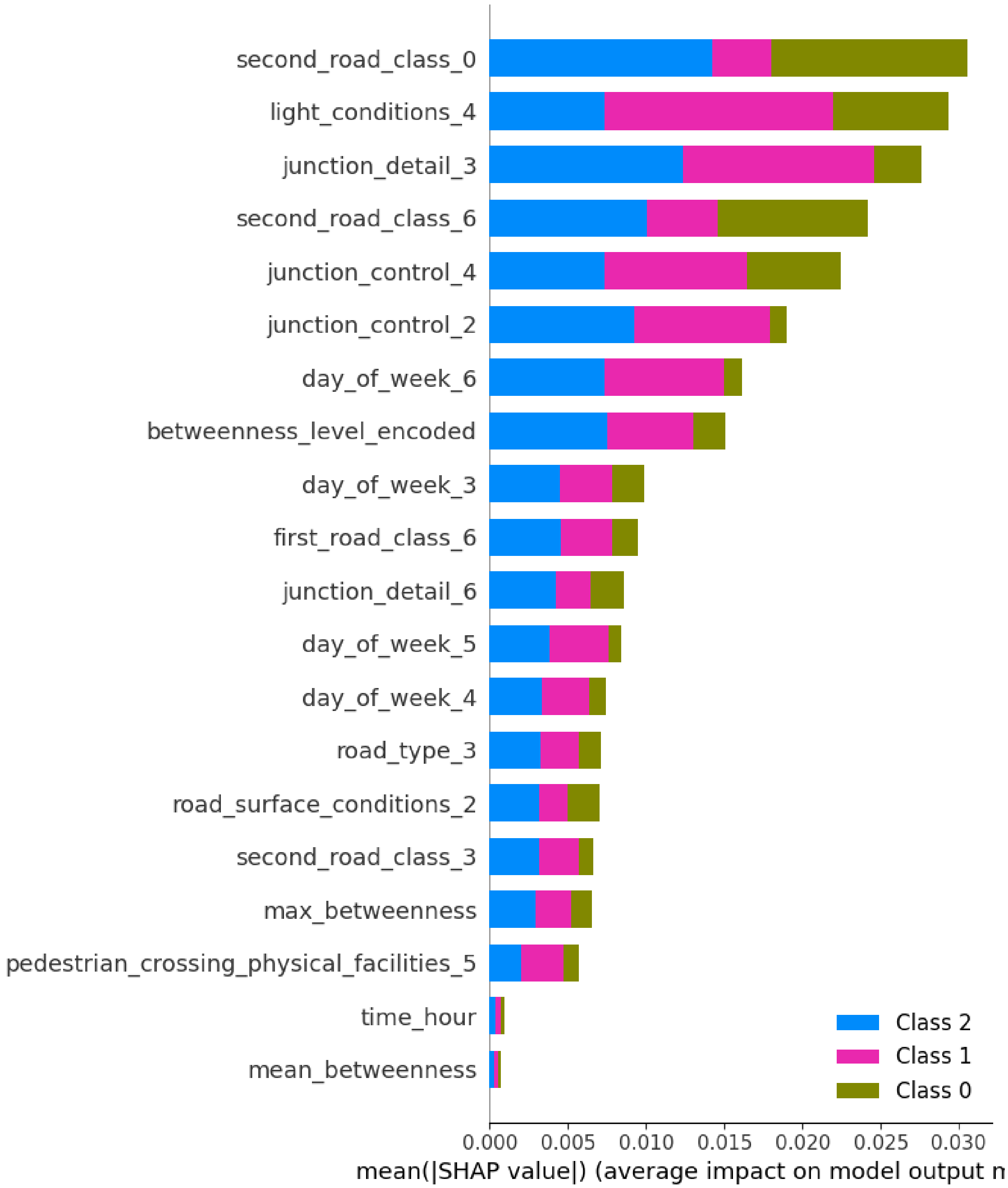
[35]:
```
import shap

#     RF   pipeline
best_pipeline_rf = grid_search_rf.best_estimator_
rf_model = best_pipeline_rf.named_steps['rf']
```

```
#
X_train_raw = X_train.copy()

#
X_sample = X_train_raw.sample(n=5000, random_state=42)

#   SHAP Explainer TreeExplainer    RF
explainer = shap.Explainer(rf_model, X_sample)
shap_values = explainer(X_sample)
```

```
100%|===================| 14940/15000 [02:07<00:00]
```

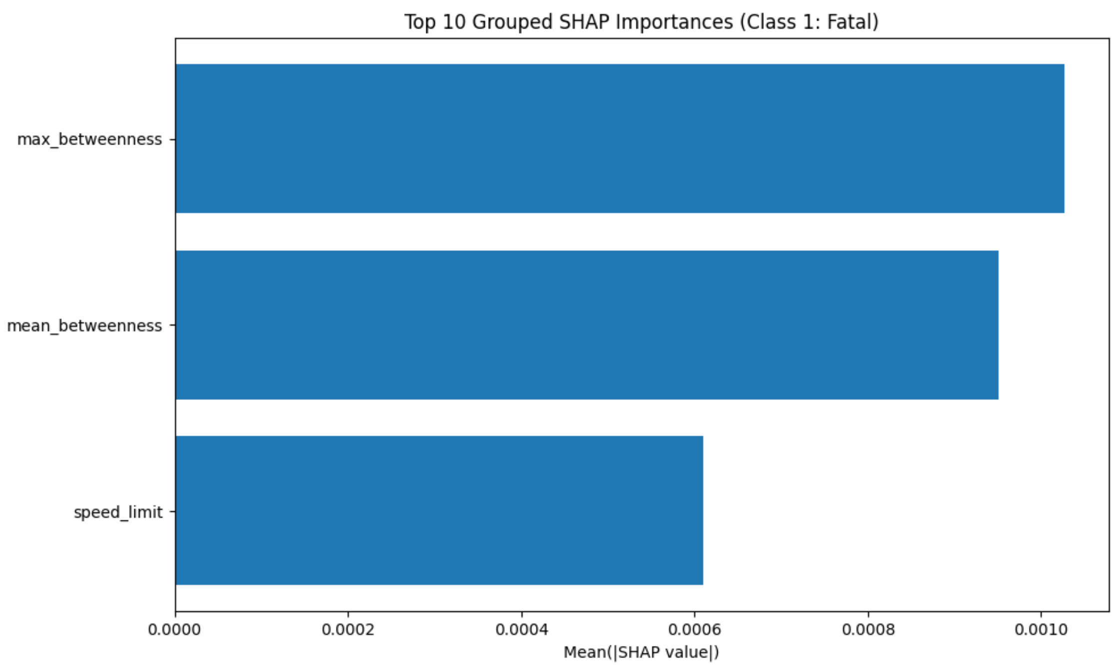
[40]:

```
# Summary plot
shap.summary_plot(shap_values, X_sample)
```



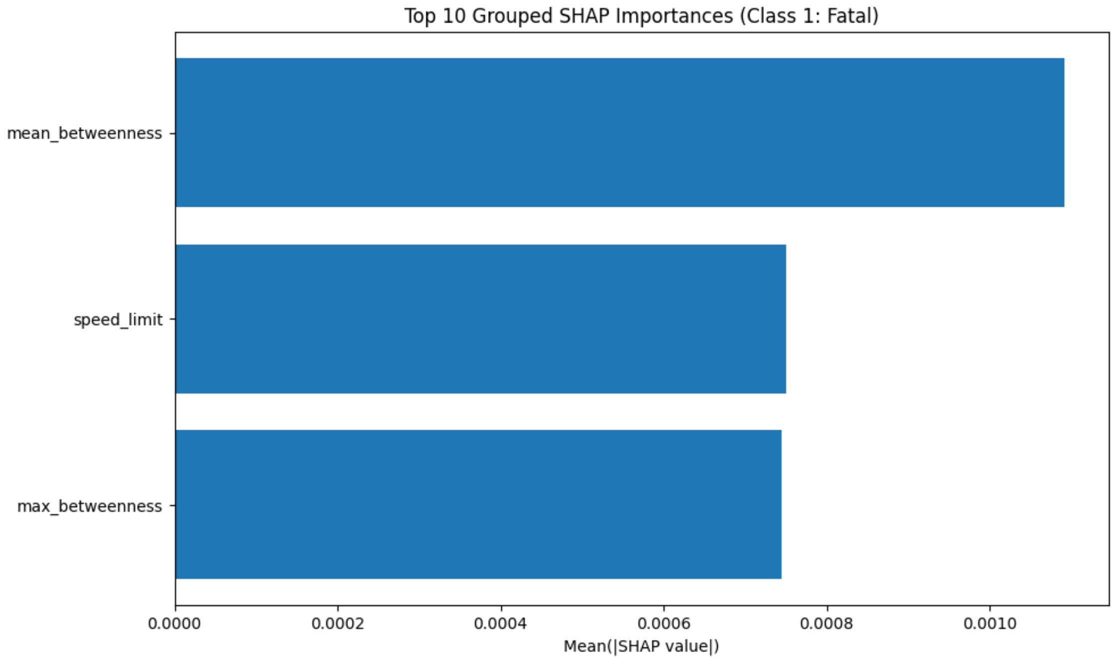
[37]:
```python
import numpy as np
import pandas as pd
from collections import defaultdict
import re
import matplotlib.pyplot as plt

#   target_class   0 / 1 / 2    fatal / serious / slight
```

```python
target_class = 0  # fatal
shap_vals = shap_values[target_class]
shap_mean_abs = np.abs(shap_vals.values).mean(axis=0)

feature_names = X_train_raw.columns
grouped_shap = defaultdict(float)

for feat_name, shap_val in zip(feature_names, shap_mean_abs):
    match = re.match(r"(.+?)_(\d+)$", feat_name)
    if match:
        base_feat = match.group(1)
    else:
        base_feat = feat_name
    grouped_shap[base_feat] += shap_val

shap_df = pd.DataFrame({
    'Feature Group': list(grouped_shap.keys()),
    'Mean(|SHAP|)': list(grouped_shap.values())
}).sort_values(by='Mean(|SHAP|)', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(shap_df['Feature Group'][:10][::-1], shap_df['Mean(|SHAP|)'][:10][::
 ↪-1])
plt.xlabel("Mean(|SHAP value|)")
plt.title("Top 10 Grouped SHAP Importances (Class 1: Fatal)")
plt.tight_layout()
plt.show()
```



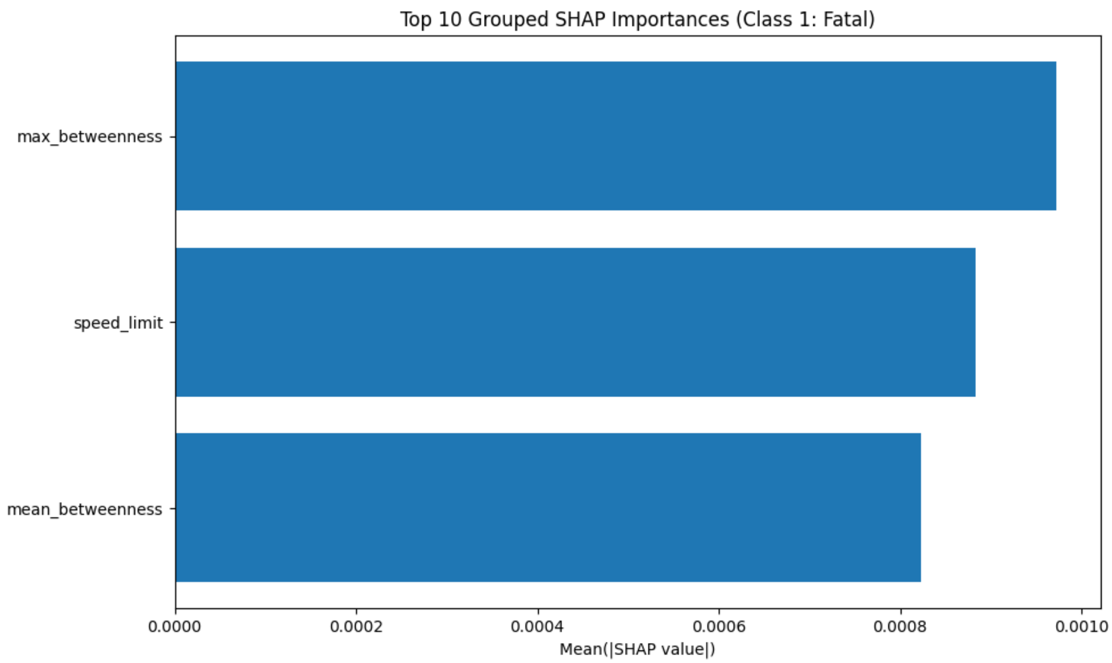
[38]:

```
#   target_class   0 / 1 / 2    fatal / serious / slight
target_class = 1  # fatal
shap_vals = shap_values[target_class]
shap_mean_abs = np.abs(shap_vals.values).mean(axis=0)

feature_names = X_train_raw.columns
grouped_shap = defaultdict(float)

for feat_name, shap_val in zip(feature_names, shap_mean_abs):
    match = re.match(r"(.+?)_(\d+)$", feat_name)
    if match:
        base_feat = match.group(1)
    else:
        base_feat = feat_name
    grouped_shap[base_feat] += shap_val

shap_df = pd.DataFrame({
    'Feature Group': list(grouped_shap.keys()),
    'Mean(|SHAP|)': list(grouped_shap.values())
}).sort_values(by='Mean(|SHAP|)', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(shap_df['Feature Group'][:10][::-1], shap_df['Mean(|SHAP|)'][:10][::
 ↪-1])
plt.xlabel("Mean(|SHAP value|)")
plt.title("Top 10 Grouped SHAP Importances (Class 1: Fatal)")
plt.tight_layout()
plt.show()
```



[39]:
```
#   target_class   0 / 1 / 2    fatal / serious / slight
target_class = 2  # fatal
shap_vals = shap_values[target_class]
shap_mean_abs = np.abs(shap_vals.values).mean(axis=0)

feature_names = X_train_raw.columns
grouped_shap = defaultdict(float)

for feat_name, shap_val in zip(feature_names, shap_mean_abs):
    match = re.match(r"(.+?)_(\d+)$", feat_name)
    if match:
        base_feat = match.group(1)
    else:
        base_feat = feat_name
    grouped_shap[base_feat] += shap_val

shap_df = pd.DataFrame({
    'Feature Group': list(grouped_shap.keys()),
    'Mean(|SHAP|)': list(grouped_shap.values())
}).sort_values(by='Mean(|SHAP|)', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(shap_df['Feature Group'][:10][::-1], shap_df['Mean(|SHAP|)'][:10][::
 ↪-1])
plt.xlabel("Mean(|SHAP value|)")
```

```
plt.title("Top 10 Grouped SHAP Importances (Class 1: Fatal)")
plt.tight_layout()
plt.show()
```



2.3 Results and discussion

[go back to the top]

Three supervised learning models were trained to classify accident severity: **Logistic Regression**, **Random Forest**, and **XGBoost**, using a dataset of **115,805 records** from 2015–2019 with **97 numerical features**. The target variable had a highly imbalanced distribution: **"slight" = 87.6%**, **"serious" = 11.9%**, and **"fatal" = 0.5%**, making **macro-F1 and per-class recall** more appropriate than accuracy for evaluation.

- **Logistic Regression** achieved the highest overall accuracy (**0.88**), but failed to detect any "fatal" or "serious" cases (macro-F1 = **0.31**).

- **XGBoost** offered slightly better recall for the "serious" class and achieved a macro-F1 of **0.32**, but remained heavily biased toward the majority class.

- **Random Forest** delivered the most balanced performance, with an accuracy of **0.81** and a macro-F1 score of **0.35**, successfully identifying a subset of minority cases (recall: fatal = 0.02, serious = 0.08).

These results illustrate the risk of relying on accuracy in imbalanced classification. For example, **Dandibhotla et al. (2022)** reported 96.18% accuracy using XGBoost, but did not address class imbalance or report recall, making such models less reliable in safety-critical applications. This study emphasizes metrics that reflect model fairness across all classes.

Random Forest's feature importance revealed that **time_hour, day_of_week, and max_betweenness** were consistently impactful. Max_betweenness, a spatial indicator derived from borough-level road network topology, appeared among the top 5 predictors, validating the integration of spatial structure into severity modeling.

To further interpret the model, **SHAP (SHapley Additive Explanations)** was applied. Global SHAP analysis confirmed the high contribution of temporal and spatial features. Class-specific SHAP bar plots showed that: - For **fatal** accidents, the most influential features were max_betweenness, speed_limit, and light_conditions_5 (dark, no street lighting). - For **serious** accidents, junction_detail, pedestrian_crossing_physical_facilities, and first_road_class played larger roles. - For **slight** accidents, time_hour, weather_conditions, and general road context were dominant.

These findings demonstrate that combining **spatial network metrics** with **contextual accident features** enhances both predictive accuracy and model interpretability for road safety applications.

2.4 Conclusion

[go back to the top]

This study investigated whether supervised machine learning models can accurately predict the severity of road traffic accidents in London, using spatial, temporal, and environmental features. A borough-level dataset from 2015 to 2019 was constructed, integrating UK accident records with spatial centrality indicators derived from OpenStreetMap.

Among the models tested, Random Forest demonstrated the best overall balance between performance and interpretability, achieving a macro-F1 score of 0.35. SHAP analysis further revealed that features such as max_betweenness, time_hour, and road type significantly contributed to severity predictions. These results suggest that incorporating spatial network metrics meaningfully enhances the capacity of data-driven safety models to identify severe accident risks.

However, this project has several limitations. The severe class imbalance in the dataset limited the model's ability to generalize predictions for rare fatal cases. The spatial resolution was restricted to the borough level, which may obscure finer-scale local effects. In addition, the study only used tabular features; incorporating trajectory-level or vehicle-specific data could improve model fidelity.

Future work may explore finer spatial units, additional data modalities (e.g., street view imagery or traffic flow), and deep learning methods for enhanced prediction and interpretability.

[ ]:

2.5 References

[go back to the top]

[ ]: