What factors significantly influence road accident severity in London, and how can these factors be effectively utilized to predict accident severity levels?

Preparation

- Github link
- Number of words: 1479
- Runtime: 2 minutes (*Memory 16 GB, CPU Intel i9-14900*)
- Coding environment: SDS Docker
-

Table of contents

1 Introduction

[go back to the top]

Road traffic accidents remain a significant public health and safety concern in the United Kingdom, particularly in densely populated urban areas like London. Understanding the factors that influence accident severity is crucial for developing targeted strategies to reduce serious and fatal accidents. Recent advancements in data analytics and machine learning have created new opportunities for analyzing road accident data with greater precision. These techniques allow researchers to identify complex patterns and relationships that might be missed by traditional statistical approaches (Behboudi et al., 2023). Wang et al. (2013) highlighted the importance of road characteristics, traffic features, environmental conditions, and human factors. Their review demonstrated that road type and geometric design significantly influence accident outcomes, with certain configurations associated with more severe incidents. Focusing specifically on motorway accidents in England, Michalaki et al. (2015) employed generalized ordered logistic regression models to identify risk factors. Their findings revealed that accidents occurring in darkness, on curved segments, involving motorcycles, or during adverse weather conditions were more likely to result in severe outcomes. In the realm of methodological approaches, AlHashmi (2023) demonstrated the value of machine learning models in accident severity prediction. His comparative analysis of various algorithms showed that ensemble methods like Random Forest often outperform traditional statistical models by capturing non-linear relationships and complex interactions between variables. Despite this substantial body of research, there remains a need for London-specific analysis using recent data and contemporary methodological approaches. This study addresses this gap by analyzing the 2023 road accident data for London, employing Multinomial Logistic Regression models and Random Forest to identify key determinants of accident severity and develop predictive models.

2 Research questions

[go back to the top]

This study addresses the following primary research question:

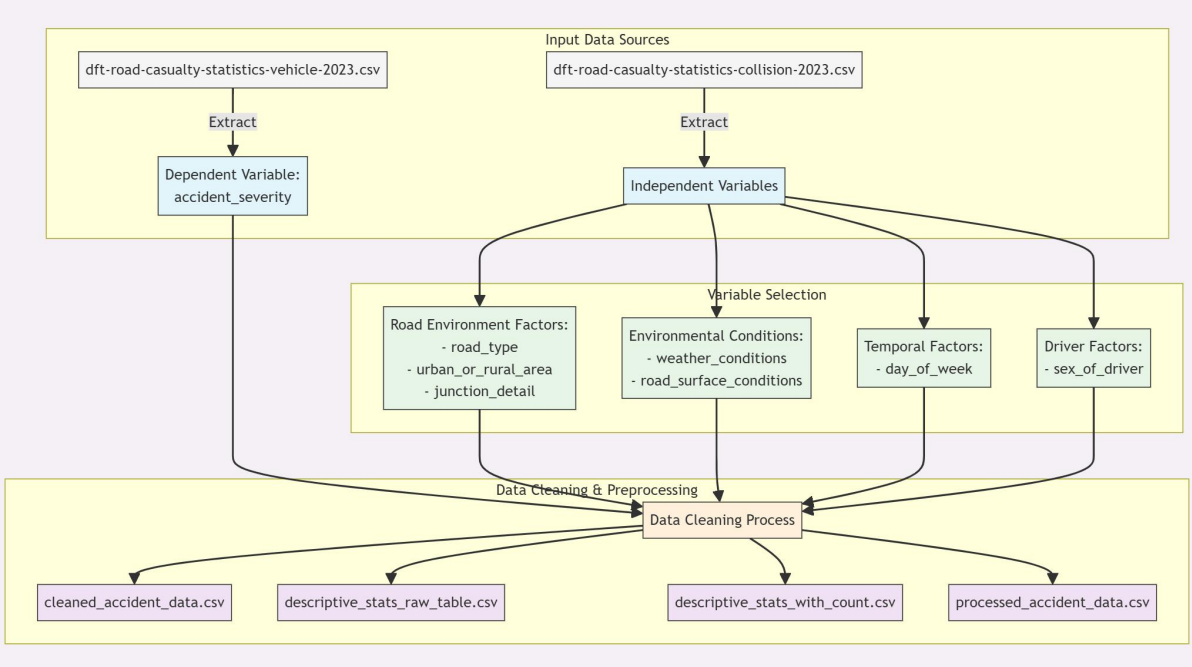
What factors significantly influence road accident severity in London, and how can these factors be effectively utilized to predict accident severity levels?

To comprehensively address this main question, the following sub-questions will be investigated:

- Which factors (road type, weather conditions, lighting conditions, etc.) are most strongly associated with different levels of accident severity in London?
- To what extent can Multinomial Logistic Regression models and Random Forest algorithm accurately predict accident severity based on identified significant factors?

3 Data

[go back to the top]



This research utilizes the Road Safety Data provided by the UK government.

Data source link: UK Government Road Safety Data

The dataset used in this Notebook primarily consists of two related tables:

dft-road-casualty-statistics-collision-2023.csv

dft-road-casualty-statistics-vehicle-2023.csv

3.1 Data Loading and Initial Exploration

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')

# Set display options
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', 20)
plt.style.use('ggplot')

# Load data
collision_data_url = 'https://raw.githubusercontent.com/EvanFeng11/CASA06_Assessment/main/data/dft-road-casualty-statistic
vehicle_data_url = 'https://raw.githubusercontent.com/EvanFeng11/CASA06_Assessment/main/data/dft-road-casualty-statistics-

collision_data = pd.read_csv(collision_data_url)
vehicle_data = pd.read_csv(vehicle_data_url)
```

3.2 Variable Selection and Understanding

Dependent Variable:

- `accident_severity`: Severity of the accident (1=Fatal, 2=Serious, 3=Slight)

Independent Variables: Road Environment Factors:

- `road_type` : Type of road (e.g., Motorway, A road, Unclassified road)
- `urban_or_rural_area` : Urban or rural area (1=Urban, 2=Rural)
- `junction_detail` : Junction details (describing the type of junction where the accident occurred)

Environmental Condition Factors:

- `weather_conditions` : Weather conditions (e.g., Fine, Rain, Fog)
- `road_surface_conditions` : Road surface conditions (e.g., Dry, Wet/Damp, Ice)

Temporal Factors:

- `day_of_week` : Day of the week (1-7 representing Sunday to Saturday)

Driver Factors:

- `sex_of_driver` : Driver's gender (1=Male, 2=Female)

3.3 Data Cleaning and Preprocessing

In [2]:
```
# Select needed columns
cols_collision = ['accident_index', 'accident_severity', 'day_of_week',
                  'road_type', 'urban_or_rural_area', 'junction_detail',
                  'weather_conditions', 'road_surface_conditions']

cols_vehicle = ['accident_index', 'vehicle_reference', 'sex_of_driver']

# Extract needed data
collision_selected = collision_data[cols_collision]
vehicle_selected = vehicle_data[cols_vehicle]

# Handle missing values
# For categorical variables, we can fill with the most frequent value or create an "Unknown" category
collision_selected['road_type'].fillna(-1, inplace=True)  # -1 represents unknown
collision_selected['junction_detail'].fillna(-1, inplace=True)
collision_selected['weather_conditions'].fillna(-1, inplace=True)
collision_selected['road_surface_conditions'].fillna(-1, inplace=True)

# Check missing values after processing
print("\nMissing values in collision dataset after processing:")
print(collision_selected.isnull().sum())

# Convert accident severity to ordinal categories
severity_mapping = {1: 'Fatal', 2: 'Serious', 3: 'Slight'}
collision_selected['severity_category'] = collision_selected['accident_severity'].map(severity_mapping)

# Check unique values for categorical variables
for col in ['road_type', 'urban_or_rural_area', 'junction_detail',
            'weather_conditions', 'road_surface_conditions', 'day_of_week']:
    print(f"\nUnique values for {col}:")
    print(collision_selected[col].value_counts())

# Check unique values for driver gender
print("\nUnique values for sex_of_driver:")
print(vehicle_selected['sex_of_driver'].value_counts())
```

```
Missing values in collision dataset after processing:
accident_index             0
accident_severity          0
day_of_week                0
road_type                  0
urban_or_rural_area        0
junction_detail            0
weather_conditions         0
road_surface_conditions    0
dtype: int64

Unique values for road_type:
road_type
6    75685
3    15412
1     5947
9     3177
2     2236
7     1801
Name: count, dtype: int64

Unique values for urban_or_rural_area:
urban_or_rural_area
 1    70312
 2    33934
-1        8
 3        4
Name: count, dtype: int64

Unique values for junction_detail:
junction_detail
 0     42978
 3     29039
 6      9507
 1      7545
 9      6183
 99     2397
 8      2263
 7      1634
 2      1583
 5      1128
-1         1
Name: count, dtype: int64

Unique values for weather_conditions:
weather_conditions
1    82029
2    12840
9     3335
8     3081
5     1225
4      977
3      396
7      333
6       42
Name: count, dtype: int64

Unique values for road_surface_conditions:
road_surface_conditions
 1    72752
 2    26944
 9     1617
 4     1461
-1     1064
 3      241
 5      179
Name: count, dtype: int64

Unique values for day_of_week:
day_of_week
6    17118
5    15791
4    15749
3    15431
2    14288
7    14182
1    11699
Name: count, dtype: int64

Unique values for sex_of_driver:
sex_of_driver
1    115739
2     48528
3     25548
Name: count, dtype: int64
```

In [3]:
```
# Merge datasets
# Only keep the record of the first driver (suitable for analysis focusing on accidents rather than vehicles)
# First, select the first vehicle for each accident
first_vehicle = vehicle_selected.sort_values(['accident_index', 'vehicle_reference']).drop_duplicates('accident_index')
merged_first = pd.merge(collision_selected, first_vehicle, on='accident_index', how='inner')

# delete vehicle_reference
merged_first = merged_first.drop(columns=['vehicle_reference'])

# View the merged dataset

print("\nMerged dataset (first vehicle):")
print(f"Rows: {merged_first.shape[0]}, Columns: {merged_first.shape[1]}")
display(merged_first.head())

# Save the processed dataset
# merged_first.to_csv('data/processed_accident_data.csv', index=False)
```

```
Merged dataset (first vehicle):
Rows: 90580, Columns: 10
```

	accident_index	accident_severity	day_of_week	road_type	urban_or_rural_area	junction_detail	weather_conditions	road_sur
0	2023010419171	3	1	2	1	9	8	
1	2023010419183	3	1	6	1	3	1	
2	2023010419189	3	1	1	1	1	1	
3	2023010419191	3	1	6	1	3	9	
4	2023010419192	3	1	6	1	8	1	

In [4]:
```
import pandas as pd

# Read the data link

accident_data = pd.read_csv('https://raw.githubusercontent.com/EvanFeng11/CASA06_Assessment/main/data/processed_accident_d


# Define values to be removed and their corresponding columns
values_to_remove = {
    'junction_detail': [-1, 9, 99],
    'road_surface_conditions': [-1, 9],
    'road_type': [9],
    'sex_of_driver': [3],
    'urban_or_rural_area': [-1, 3],
    'weather_conditions': [8, 9]


}

# Create filter condition
mask = True
for column, values in values_to_remove.items():
    for value in values:
        mask = mask & (accident_data[column] != value)

# Apply filter condition
cleaned_data = accident_data[mask]

# Display row counts before and after cleaning
print(f"Original data row count: {len(accident_data)}")
print(f"Cleaned data row count: {len(cleaned_data)}")

# Save cleaned data
# cleaned_data.to_csv('data/cleaned_accident_data.csv', index=False)

# Display first few rows of cleaned data
cleaned_data.head()
```

```
Original data row count: 90580
Cleaned data row count: 68945
```

Out[4]:

	accident_index	accident_severity	day_of_week	road_type	urban_or_rural_area	junction_detail	weather_conditions	road_su
1	2023010419183	3	1	6	1	3	1	
2	2023010419189	3	1	1	1	1	1	
4	2023010419192	3	1	6	1	8	1	
6	2023010419201	3	1	6	1	6	1	
7	2023010419205	3	1	3	2	0	1	

In [5]:
```
# Import necessary libraries
import pandas as pd
```

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

# Define variable mappings based on the data guide
variable_mappings = {
    'day_of_week': {
        1: 'Sunday',
        2: 'Monday',
        3: 'Tuesday',
        4: 'Wednesday',
        5: 'Thursday',
        6: 'Friday',
        7: 'Saturday'
    },
    'accident_severity': {
        1: 'Fatal',
        2: 'Serious',
        3: 'Slight'
    },
    'road_type': {
        1: 'Roundabout',
        2: 'One way street',
        3: 'Dual carriageway',
        6: 'Single carriageway',
        7: 'Slip road',

    },
    'urban_or_rural_area': {
        1: 'Urban',
        2: 'Rural',

    },
    'junction_detail': {
        0: 'Not at junction or within 20 metres',
        1: 'Roundabout',
        2: 'Mini-roundabout',
        3: 'T or staggered junction',
        4: 'Slip road',
        5: 'Crossroads',
        6: 'More than 4 arms (not roundabout)',
        7: 'Private drive or entrance',
        8: 'Other junction',

    },
    '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',


    },
    'road_surface_conditions': {
        1: 'Dry',
        2: 'Wet or damp',
        3: 'Snow',
        4: 'Frost or ice',
        5: 'Flood over 3cm deep',
        6: 'Oil or diesel',
        7: 'Mud',

    },
    'sex_of_driver': {
        1: 'Male',
        2: 'Female',


    }
}

# Create descriptive statistics table for independent variables by accident severity
def create_descriptive_stats_table(df, independent_vars, variable_mappings):
    """
    Create a descriptive statistics table for independent variables by accident severity.
    The sum of percentages across all values of all variables and all severity categories equals 100%.

    Parameters:
    df (DataFrame): The processed accident data
    independent_vars (list): List of independent variables to analyze
    variable_mappings (dict): Dictionary mapping variable codes to their labels
```

```
    Returns:
    DataFrame: Table with descriptive statistics
    """
    # Initialize an empty list to store results
    results = []

    # Define severity categories and their codes
    severity_mapping = {1: 'Fatal', 2: 'Serious', 3: 'Slight'}

    # Calculate total number of accidents
    total_accidents = len(df)

    # Loop through each independent variable
    for var in independent_vars:
        # Skip variables that don't exist in the dataframe
        if var not in df.columns:
            print(f"Warning: Variable '{var}' not found in the dataframe. Skipping.")
            continue

        # Get unique values for the variable
        unique_values = sorted(df[var].unique())

        # For each value, calculate the distribution across severity categories
        for value in unique_values:
            # Get the label for this value
            if var in variable_mappings and value in variable_mappings[var]:
                label = variable_mappings[var][value]
            else:
                label = f"Unknown ({value})"

            # Calculate counts for each severity category
            for severity_code, severity_name in severity_mapping.items():
                # Count accidents with this value and severity
                count = df[(df[var] == value) & (df['accident_severity'] == severity_code)].shape[0]

                # Calculate percentage of total accidents
                percentage = (count / total_accidents * 100)

                # Add row to results
                results.append({
                    'Independent Variables': var,
                    'Values': f"{value} ({label})",
                    'Severity': severity_name,
                    'Count': count,
                    'Percentage (%)': round(percentage, 2)
                })

    # Convert to DataFrame
    result_df = pd.DataFrame(results)

    # Group by independent variable to ensure they appear together
    result_df = result_df.sort_values(['Independent Variables', 'Values', 'Severity'])

    return result_df

# Define the independent variables to analyze
independent_vars = ['day_of_week', 'road_type', 'urban_or_rural_area',
                    'junction_detail', 'weather_conditions', 'road_surface_conditions',
                    'sex_of_driver']

# Create the descriptive statistics table
desc_stats_table = create_descriptive_stats_table(cleaned_data, independent_vars, variable_mappings)

# Display the table
print(" Descriptive statistics for independent variables (features) 1.")
display(desc_stats_table)

# Verify that percentages sum to 100%
total_percentage = desc_stats_table['Percentage (%)'].sum()
print(f"\nSum of all percentages: {total_percentage:.2f}%")

# Pivot the table to make it more readable
pivot_table = desc_stats_table.pivot_table(
    index=['Independent Variables', 'Values'],
    columns='Severity',
    values='Percentage (%)',
    aggfunc='sum'
).reset_index()

# Ensure all severity columns exist
for severity in ['Fatal', 'Serious', 'Slight']:
    if severity not in pivot_table.columns:
        pivot_table[severity] = 0.0

# Calculate the total percentage for each row
```

```
pivot_table['Total (%)'] = pivot_table['Fatal'] + pivot_table['Serious'] + pivot_table['Slight']

# Reorder columns
pivot_table = pivot_table[['Independent Variables', 'Values', 'Fatal', 'Serious', 'Slight', 'Total (%)']]

# Display the pivot table
print("\nPivot Table:")
display(pivot_table)

# Create a more visually appealing table with styling
styled_table = pivot_table.style.set_caption(" Descriptive statistics for independent variables (features) 2.")\
                                .set_table_styles([
                                    {'selector': 'caption',
                                     'props': [('font-weight', 'bold'),
                                               ('font-size', '1.1em'),
                                               ('text-align', 'center')]},
                                    {'selector': 'th',
                                     'props': [('font-weight', 'bold'),
                                               ('background-color', '#f2f2f2'),
                                               ('text-align', 'center')]},
                                ])\
                                .format({
                                    'Fatal': '{:.2f}',
                                    'Serious': '{:.2f}',
                                    'Slight': '{:.2f}',
                                    'Total (%)': '{:.2f}'
                                })

# Display the styled table
display(styled_table)
```

```
Descriptive statistics for independent variables (features) 1.
```

	Independent Variables	Values	Severity	Count	Percentage (%)
0	day_of_week	1 (Sunday)	Fatal	155	0.22
1	day_of_week	1 (Sunday)	Serious	2049	2.97
2	day_of_week	1 (Sunday)	Slight	5597	8.12
3	day_of_week	2 (Monday)	Fatal	178	0.26
4	day_of_week	2 (Monday)	Serious	2201	3.19
...	...	...	...	...	...
82	weather_conditions	6 (Snowing + high winds)	Serious	9	0.01
83	weather_conditions	6 (Snowing + high winds)	Slight	26	0.04
84	weather_conditions	7 (Fog or mist)	Fatal	5	0.01
85	weather_conditions	7 (Fog or mist)	Serious	47	0.07
86	weather_conditions	7 (Fog or mist)	Slight	187	0.27

108 rows × 5 columns

```
Sum of all percentages: 699.97%

Pivot Table:
```

Severity	Independent Variables	Values	Fatal	Serious	Slight	Total (%)
0	day_of_week	1 (Sunday)	0.22	2.97	8.12	11.31
1	day_of_week	2 (Monday)	0.26	3.19	10.31	13.76
2	day_of_week	3 (Tuesday)	0.20	3.32	11.25	14.77
3	day_of_week	4 (Wednesday)	0.21	3.61	11.18	15.00
4	day_of_week	5 (Thursday)	0.24	3.49	11.38	15.11
...	...	...	...	...	...	...
31	weather_conditions	3 (Snowing no high winds)	0.01	0.09	0.33	0.43
32	weather_conditions	4 (Fine + high winds)	0.03	0.27	0.68	0.98
33	weather_conditions	5 (Raining + high winds)	0.03	0.33	0.92	1.28
34	weather_conditions	6 (Snowing + high winds)	0.00	0.01	0.04	0.05
35	weather_conditions	7 (Fog or mist)	0.01	0.07	0.27	0.35

36 rows × 6 columns

Descriptive statistics for independent variables (features) 2.

Severity	Independent Variables	Values	Fatal	Serious	Slight	Total (%)
0	day_of_week	1 (Sunday)	0.22	2.97	8.12	11.31
1	day_of_week	2 (Monday)	0.26	3.19	10.31	13.76
2	day_of_week	3 (Tuesday)	0.20	3.32	11.25	14.77
3	day_of_week	4 (Wednesday)	0.21	3.61	11.18	15.00
4	day_of_week	5 (Thursday)	0.24	3.49	11.38	15.11
5	day_of_week	6 (Friday)	0.29	3.89	12.13	16.31
6	day_of_week	7 (Saturday)	0.25	3.54	9.96	13.75
7	junction_detail	0 (Not at junction or within 20 metres)	1.11	11.60	30.99	43.70
8	junction_detail	1 (Roundabout)	0.05	1.34	6.64	8.03
9	junction_detail	2 (Mini-roundabout)	0.00	0.34	1.43	1.77
10	junction_detail	3 (T or staggered junction)	0.34	7.08	22.91	30.33
11	junction_detail	5 (Crossroads)	0.03	0.27	0.99	1.29
12	junction_detail	6 (More than 4 arms (not roundabout))	0.08	2.35	7.94	10.37
13	junction_detail	7 (Private drive or entrance)	0.01	0.41	1.38	1.80
14	junction_detail	8 (Other junction)	0.03	0.62	2.03	2.68
15	road_surface_conditions	1 (Dry)	1.14	17.08	53.14	71.36
16	road_surface_conditions	2 (Wet or damp)	0.52	6.53	19.94	26.99
17	road_surface_conditions	3 (Snow)	0.00	0.07	0.18	0.25
18	road_surface_conditions	4 (Frost or ice)	0.01	0.27	0.92	1.20
19	road_surface_conditions	5 (Flood over 3cm deep)	0.01	0.06	0.13	0.20
20	road_type	1 (Roundabout)	0.04	1.08	5.13	6.25
21	road_type	2 (One way street)	0.01	0.38	1.63	2.02
22	road_type	3 (Dual carriageway)	0.37	3.49	12.26	16.12
23	road_type	6 (Single carriageway)	1.24	18.71	53.84	73.79
24	road_type	7 (Slip road)	0.02	0.34	1.45	1.81
25	sex_of_driver	1 (Male)	1.41	18.03	51.80	71.24
26	sex_of_driver	2 (Female)	0.26	5.98	22.52	28.76
27	urban_or_rural_area	1 (Urban)	0.64	14.74	50.46	65.84
28	urban_or_rural_area	2 (Rural)	1.03	9.27	23.86	34.16
29	weather_conditions	1 (Fine no high winds)	1.39	20.09	62.18	83.66
30	weather_conditions	2 (Raining no high winds)	0.21	3.15	9.89	13.25
31	weather_conditions	3 (Snowing no high winds)	0.01	0.09	0.33	0.43
32	weather_conditions	4 (Fine + high winds)	0.03	0.27	0.68	0.98
33	weather_conditions	5 (Raining + high winds)	0.03	0.33	0.92	1.28
34	weather_conditions	6 (Snowing + high winds)	0.00	0.01	0.04	0.05
35	weather_conditions	7 (Fog or mist)	0.01	0.07	0.27	0.35

In [6]:
```
# output CSV
# desc_stats_table.to_csv('data/descriptive_stats_raw_table.csv', index=False)
```

In [7]:
```
# Modify pivot table code, add Count column after Values column
def create_descriptive_stats_table_with_count(df, independent_vars, variable_mappings):
    """
    Create a descriptive statistics table with Count column placed after Values column
    """
    # Initialize an empty list to store results
    results = []

    # Define severity categories and their codes
    severity_mapping = {1: 'Fatal', 2: 'Serious', 3: 'Slight'}

    # Calculate total number of accidents
    total_accidents = len(df)

    # Iterate through each independent variable
    for var in independent_vars:
        # Skip variables that don't exist in the dataframe
```

```
        if var not in df.columns:
            print(f"Warning: Variable '{var}' not found in the dataframe. Skipping.")
            continue

        # Get unique values for the variable
        unique_values = sorted(df[var].unique())

        # For each value, calculate distribution across severity categories
        for value in unique_values:
            # Get the label for this value
            if var in variable_mappings and value in variable_mappings[var]:
                label = variable_mappings[var][value]
            else:
                label = f"Unknown ({value})"

            # Calculate total count for this value (all severity levels)
            total_count = df[df[var] == value].shape[0]

            # Calculate count and percentage for each severity category
            for severity_code, severity_name in severity_mapping.items():
                # Calculate number of accidents with this value and severity
                count = df[(df[var] == value) & (df['accident_severity'] == severity_code)].shape[0]

                # Calculate percentage of total accidents
                percentage = (count / total_accidents * 100)

                # Add row to results
                results.append({
                    'Independent Variables': var,
                    'Values': f"{value} ({label})",
                    'Count': total_count,  # Add total count
                    'Severity': severity_name,
                    'Severity_Count': count,  # Count for each severity level
                    'Percentage (%)': round(percentage, 2)
                })

    # Convert to DataFrame
    result_df = pd.DataFrame(results)

    # Group by independent variables to ensure they appear together
    result_df = result_df.sort_values(['Independent Variables', 'Values', 'Severity'])

    return result_df

# Create descriptive statistics table
desc_stats_table = create_descriptive_stats_table_with_count(cleaned_data, independent_vars, variable_mappings)

# Create pivot table with Count column
pivot_table_with_count = desc_stats_table.pivot_table(
    index=['Independent Variables', 'Values', 'Count'],  # Add Count to index
    columns='Severity',
    values=['Severity_Count', 'Percentage (%)'],
    aggfunc='sum'
).reset_index()

# Reorganize column names
pivot_table_with_count.columns = [
    f"{col[0]}_{col[1]}" if col[1] != "" else col[0]
    for col in pivot_table_with_count.columns
]

# Rearrange columns to place Count after Values, followed by percentages for each severity level
new_columns = ['Independent Variables', 'Values', 'Count']
for severity in ['Fatal', 'Serious', 'Slight']:
    new_columns.append(f'Percentage (%)_{severity}')

# Ensure all columns exist
for col in new_columns:
    if col not in pivot_table_with_count.columns and col not in ['Independent Variables', 'Values', 'Count']:
        pivot_table_with_count[col] = 0.0

# Calculate total percentage
pivot_table_with_count['Total (%)'] = pivot_table_with_count['Percentage (%)_Fatal'] + pivot_table_with_count['Percentage

# Select and order columns
pivot_table_with_count = pivot_table_with_count[new_columns + ['Total (%)']]


display(pivot_table_with_count)
```

	Independent Variables	Values	Count	Percentage (%)_Fatal	Percentage (%)_Serious	Percentage (%)_Slight	Total (%)
0	day_of_week	1 (Sunday)	7801	0.22	2.97	8.12	11.31
1	day_of_week	2 (Monday)	9486	0.26	3.19	10.31	13.76
2	day_of_week	3 (Tuesday)	10181	0.20	3.32	11.25	14.77
3	day_of_week	4 (Wednesday)	10343	0.21	3.61	11.18	15.00
4	day_of_week	5 (Thursday)	10412	0.24	3.49	11.38	15.11
...	...	...	...	...	...	...	...
31	weather_conditions	3 (Snowing no high winds)	298	0.01	0.09	0.33	0.43
32	weather_conditions	4 (Fine + high winds)	678	0.03	0.27	0.68	0.98
33	weather_conditions	5 (Raining + high winds)	885	0.03	0.33	0.92	1.28
34	weather_conditions	6 (Snowing + high winds)	35	0.00	0.01	0.04	0.05
35	weather_conditions	7 (Fog or mist)	239	0.01	0.07	0.27	0.35

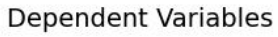
36 rows × 7 columns

In [1]:
```
# output CSV
# pivot_table_with_count.to_csv('data/descriptive_stats_with_count.csv', index=False)
```

In [9]:
```
# Create a table for dependent variable using cleaned_data dataset

# Assuming cleaned_data already exists and contains the accident_severity column
# Calculate counts for each severity level
severity_counts = cleaned_data['accident_severity'].value_counts().sort_index()

# Calculate total number of rows
total_rows = len(cleaned_data)

# Create severity mapping
severity_mapping = {1: 'Fatal', 2: 'Serious', 3: 'Slight'}

# Create data dictionary
data = {
    'Variables': [severity_mapping.get(i, f'Unknown ({i})') for i in severity_counts.index],
    'Count': severity_counts.values,
    'Percentage %': [(count / total_rows * 100) for count in severity_counts.values]
}

# Create DataFrame
df_dependent = pd.DataFrame(data)

# Set table style
pd.set_option('display.max_columns', None)
pd.set_option('display.width', None)
pd.set_option('display.float_format', '{:.2f}'.format)

# Display table
print("Dependent Variables")
print("="*60)
print("accident_severity")
print("-"*60)
display(df_dependent)

# Create visualization table
fig, ax = plt.subplots(figsize=(10, 4))
ax.axis('tight')
ax.axis('off')
table = ax.table(cellText=df_dependent.values,
                 colLabels=df_dependent.columns,
                 loc='center',
                 cellLoc='center')
table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1.2, 1.5)


plt.title('Dependent Variables', fontsize=14, pad=10)
plt.show()

# Print totals
print(f"Total count: {total_rows}")
print(f"Total percentage: 100.00%")
```

```
Dependent Variables
============================================================
accident_severity
------------------------------------------------------------
```

	Variables	Count	Percentage %
0	Fatal	1154	1.67
1	Serious	16554	24.01
2	Slight	51237	74.32



Variables	Count	Percentage %
Fatal	1154	1.6737979548915802
Serious	16554	24.01044310682428
Slight	51237	74.31575893828413

```
Total count: 68945
Total percentage: 100.00%
```

3.4 Data Preprocessing Summary

The final dataset (**cleaned_accident_data.csv**) contains 68945 records and 8 variables.

During the entire data cleaning process, three other data tables can be obtained, namely:**descriptive_stats_raw_table.csv, descriptive_stats_with_count.csv and processed_accident_data.csv.**

4 Methodology

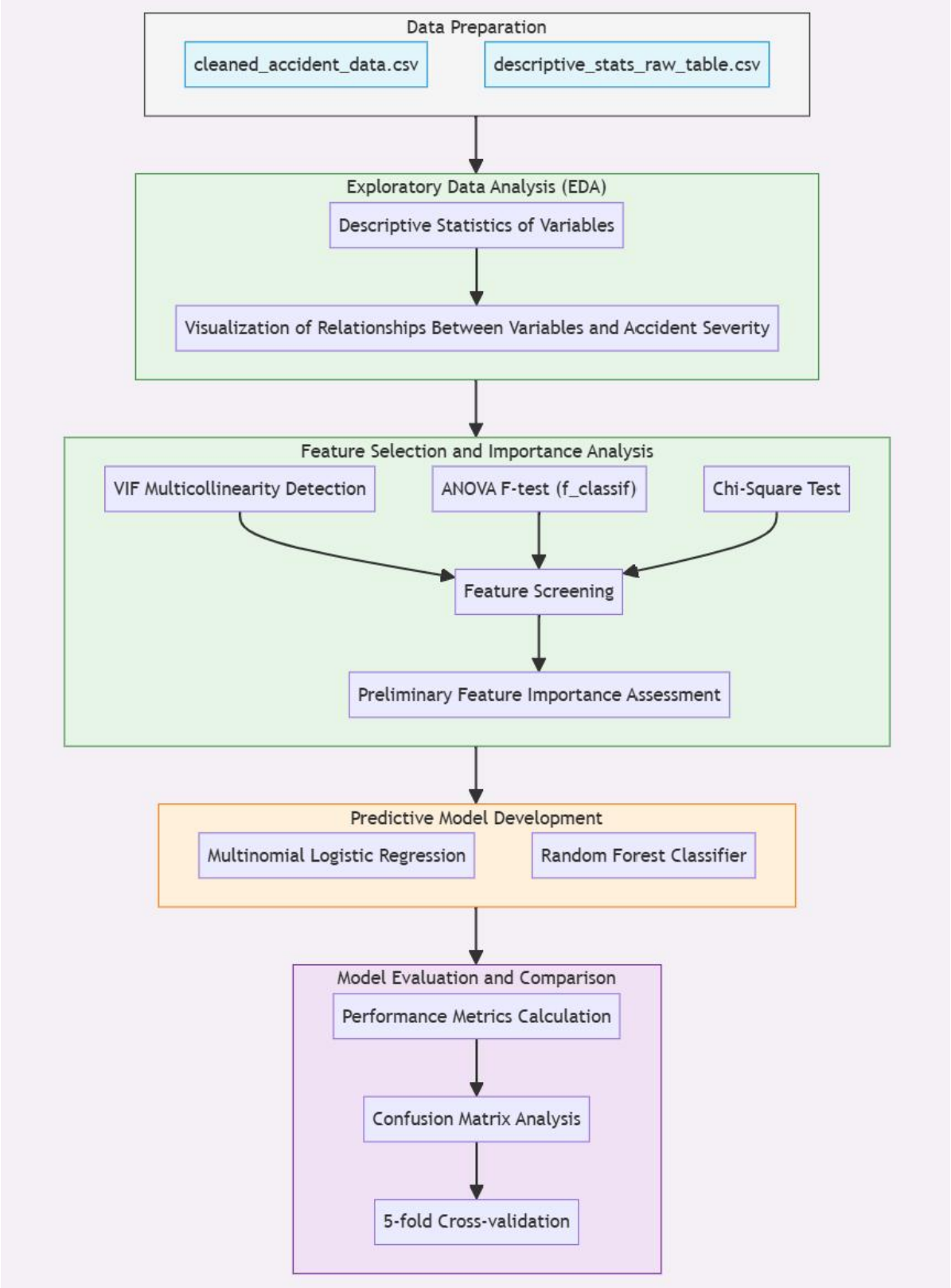
[go back to the top]

Two primary datasets were prepared for analysis:

- `cleaned_accident_data.csv`
- `descriptive_stats_raw_table.csv`

Two distinct modeling approaches were implemented to predict accident severity:

- **Multinomial Logistic Regression** : estimate the probability of different severity outcomes based on predictor variables. This model is particularly suitable for our analysis as it can handle a categorical dependent variable with more than two classes. The coefficients from this model offer direct insights into how each factor influences the likelihood of different accident severity outcomes.
- **Random Forest Classifier** : capture complex patterns in the data. This ensemble method excels at modeling non-linear relationships and interactions between variables without explicit specification, remains robust to outliers, provides alternative measures of feature importance, and typically performs well with imbalanced datasets like ours where fatal accidents are relatively rare.



5 Results and discussion

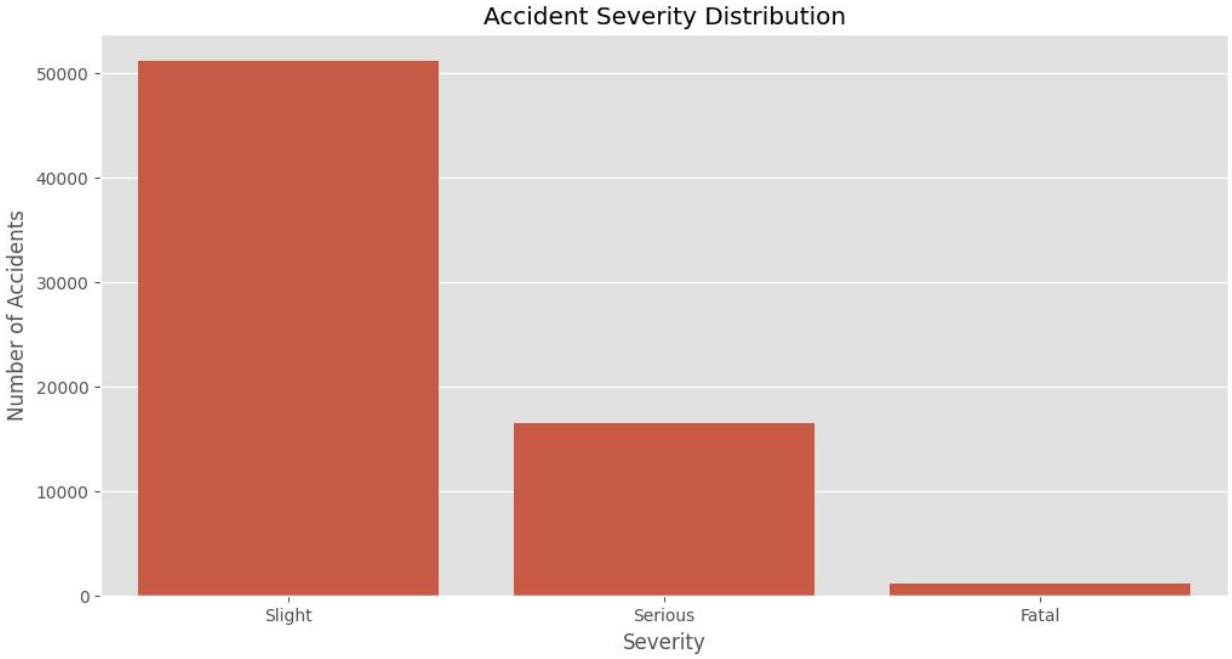
[go back to the top]

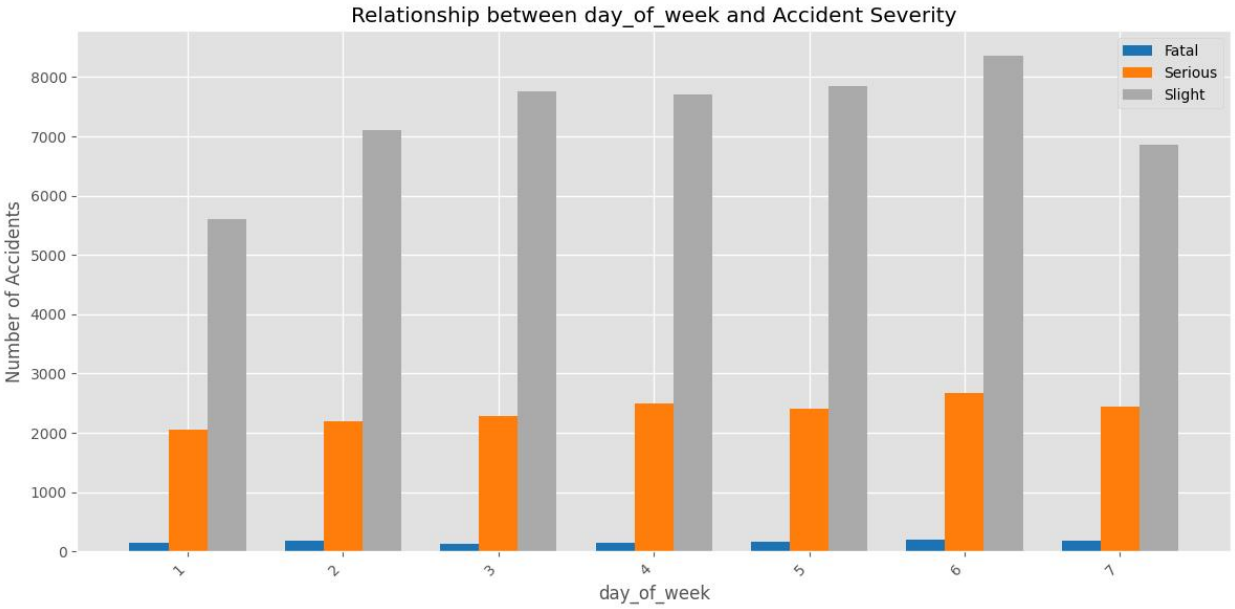
5.1 Exploratory Data Analysis (EDA)

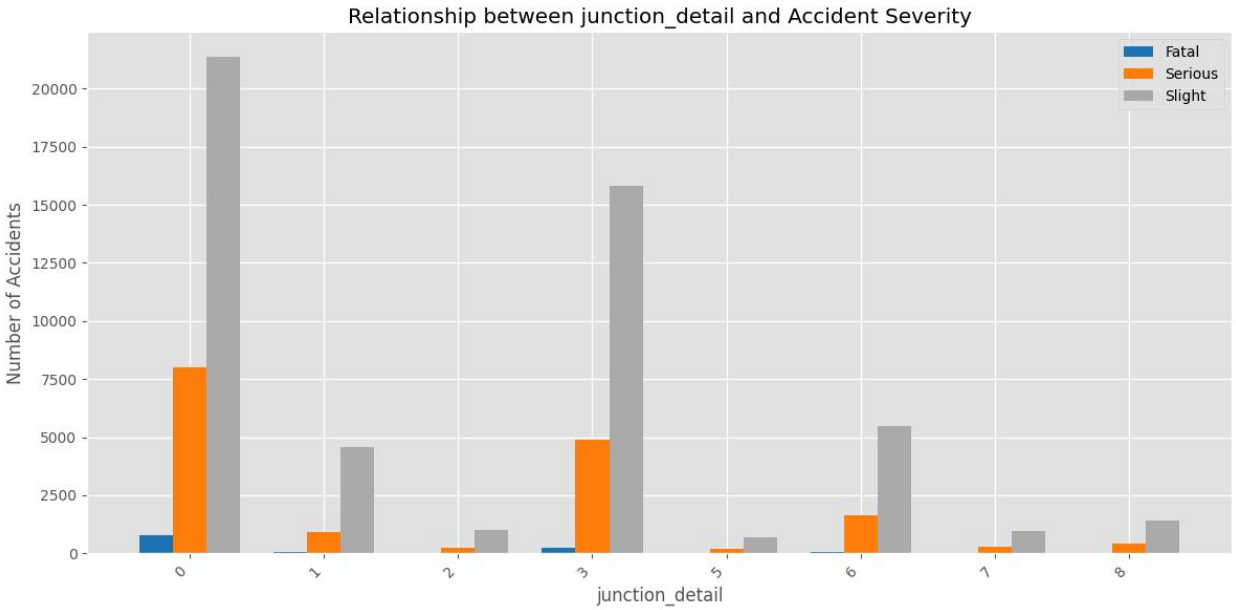
In [10]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('https://raw.githubusercontent.com/EvanFeng11/CASA06_Assessment/main/data/cleaned_accident_data.csv')
```

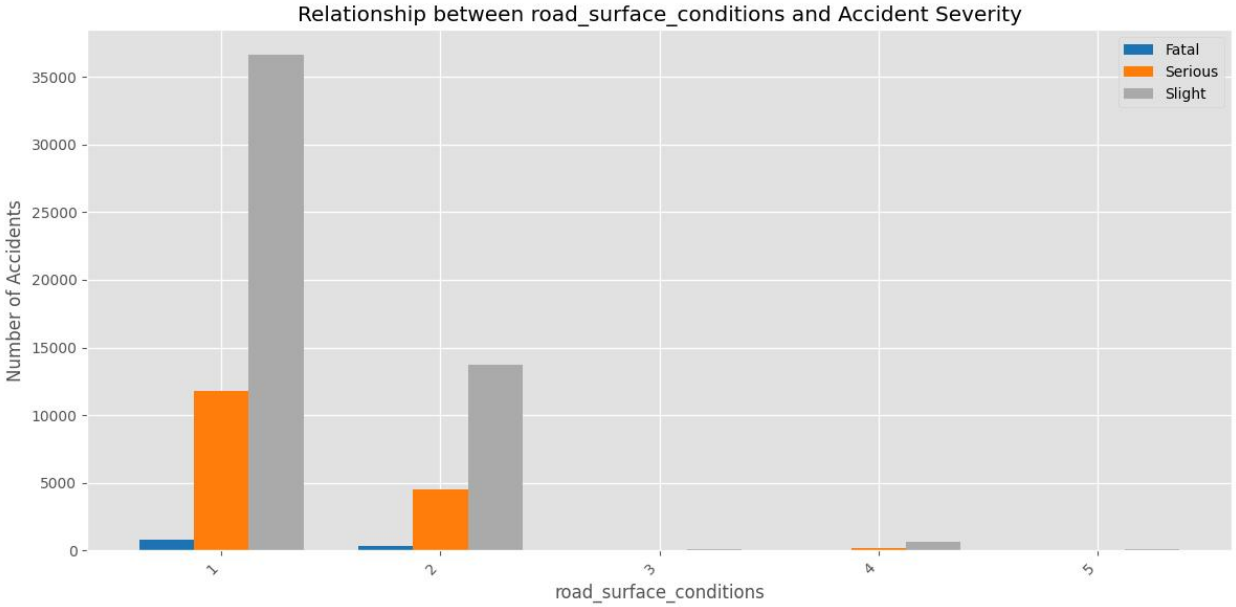
```
desc_stats_table = pd.read_csv('https://raw.githubusercontent.com/EvanFeng11/CASA06_Assessment/main/data/descriptive_stats


# Descriptive statistics
print("Accident severity distribution:")
severity_counts = df['severity_category'].value_counts()
print(severity_counts)
print(f"Severity percentages: {severity_counts / len(df) * 100}")

# Visualize severity distribution
plt.figure(figsize=(12, 6))
sns.countplot(data=df, x='severity_category')
plt.title('Accident Severity Distribution')
plt.xlabel('Severity')
plt.ylabel('Number of Accidents')
plt.show()

# Visualize relationship between variables and accident severity
# Using desc_stats_table to create bar charts
independent_vars = desc_stats_table['Independent Variables'].unique()

for var in independent_vars:
    # Filter data for current variable
    var_data = desc_stats_table[desc_stats_table['Independent Variables'] == var]

    # Get all unique values
    values = var_data['Values'].unique()

    # Prepare data for each severity level
    fatal_counts = []
    serious_counts = []
    slight_counts = []

    for val in values:
        # Filter data for specific value
        val_data = var_data[var_data['Values'] == val]

        # Get count for each severity level
        fatal = val_data[val_data['Severity'] == 'Fatal']['Count'].values[0] if 'Fatal' in val_data['Severity'].values els
        serious = val_data[val_data['Severity'] == 'Serious']['Count'].values[0] if 'Serious' in val_data['Severity'].valu
        slight = val_data[val_data['Severity'] == 'Slight']['Count'].values[0] if 'Slight' in val_data['Severity'].values

        fatal_counts.append(fatal)
        serious_counts.append(serious)
        slight_counts.append(slight)

    # Create figure
    plt.figure(figsize=(12, 6))

    # Set bar positions
    x = np.arange(len(values))
    width = 0.25

    # Create bars
    plt.bar(x - width, fatal_counts, width, label='Fatal', color='#1f77b4')
    plt.bar(x, serious_counts, width, label='Serious', color='#ff7f0e')
    plt.bar(x + width, slight_counts, width, label='Slight', color='#aaaaaa')

    # Add labels and title
    plt.xlabel(var)
    plt.ylabel('Number of Accidents')
    plt.title(f'Relationship between {var} and Accident Severity')

    # Add tick labels
    value_labels = [val.split(' (')[0] if ' (' in val else val for val in values]
    plt.xticks(x, value_labels, rotation=45, ha='right')

    # Add legend
    plt.legend()

    # Adjust layout
    plt.tight_layout()

    # Show figure
    plt.show()

# Variable correlation analysis
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
correlation = df[numeric_cols].corr()

plt.figure(figsize=(12, 10))
sns.heatmap(correlation, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Variable Correlation Heatmap')
plt.tight_layout()
plt.show()
```

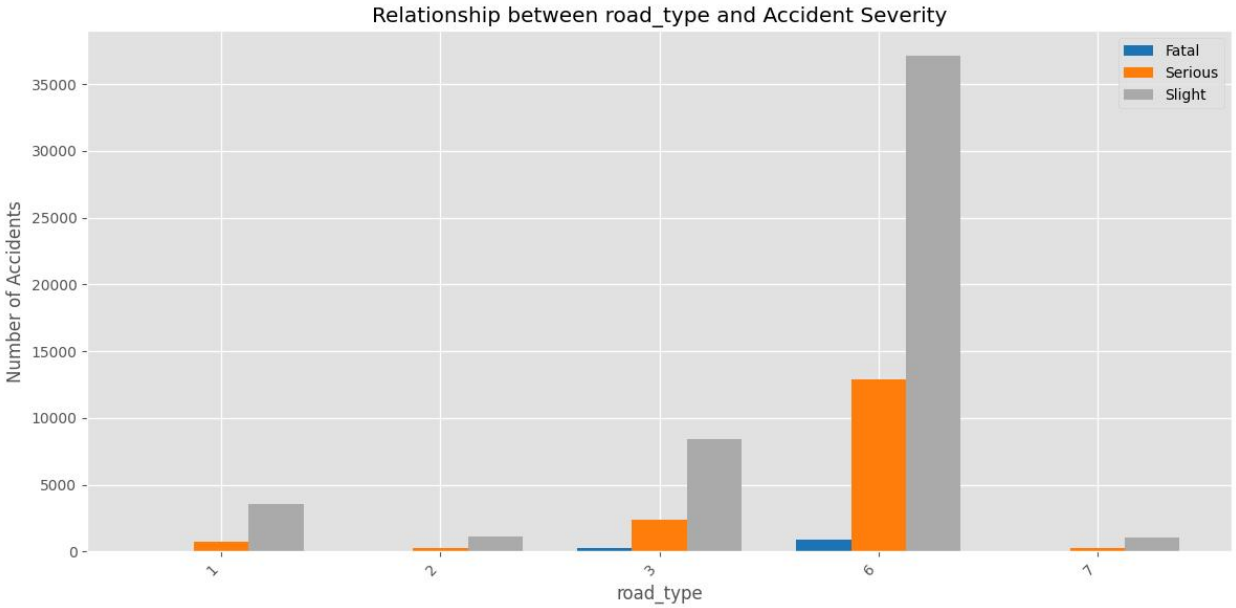
```
Accident severity distribution:
severity_category
Slight     51237
Serious    16554
Fatal       1154
Name: count, dtype: int64
Severity percentages: severity_category
Slight    74.32
Serious   24.01
Fatal      1.67
Name: count, dtype: float64
```

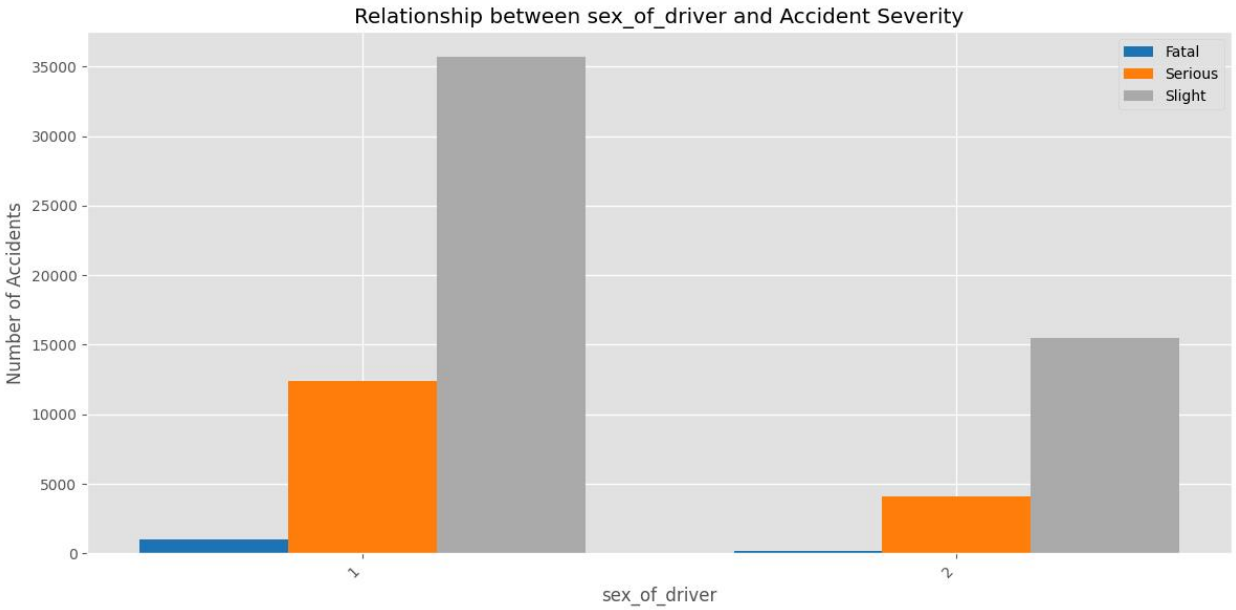


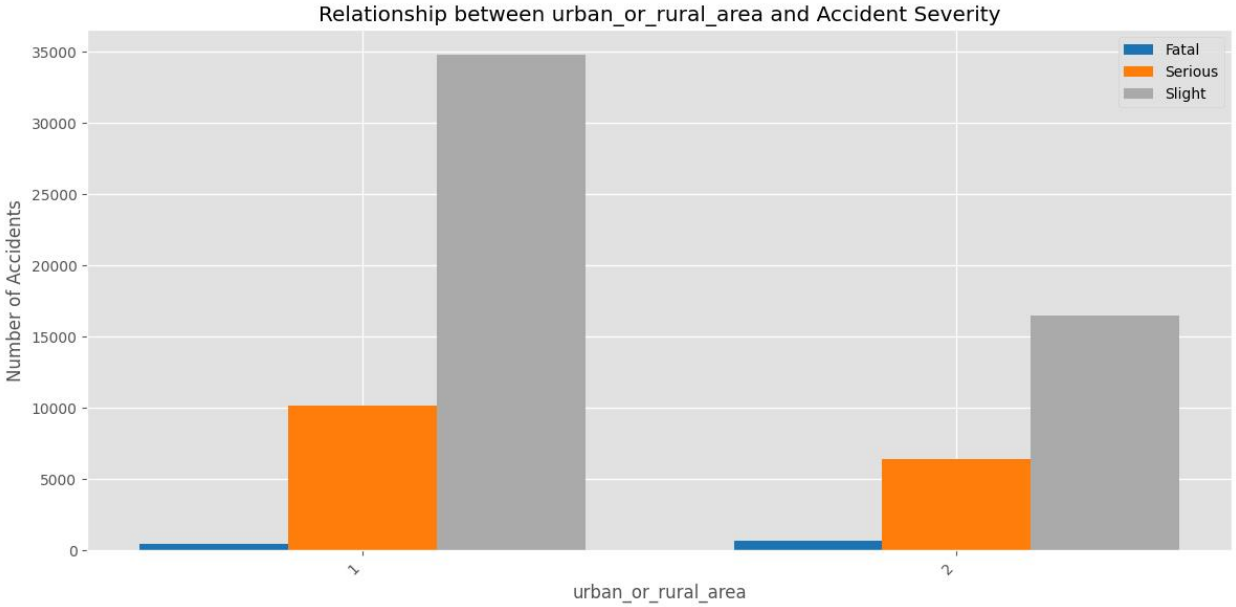


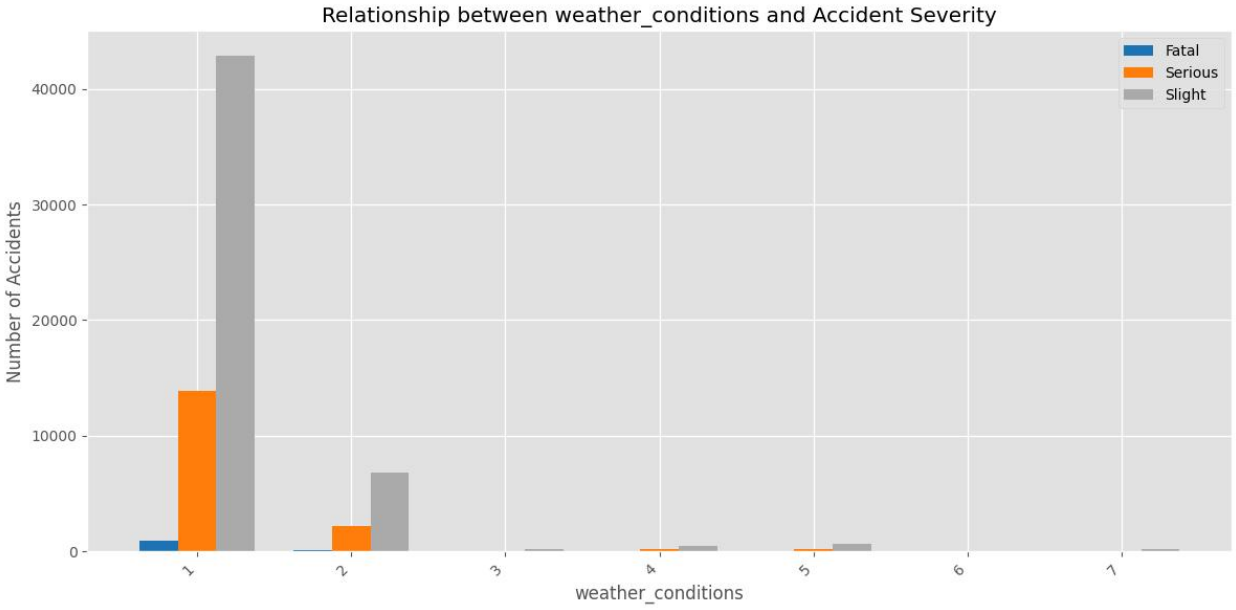

Relationship between junction_detail and Accident Severity
Fatal
Serious
Slight
Number of Accidents
0
2500
5000
7500
10000
12500
15000
17500
20000
0
1
2
3
5
6
7
8
junction_detail

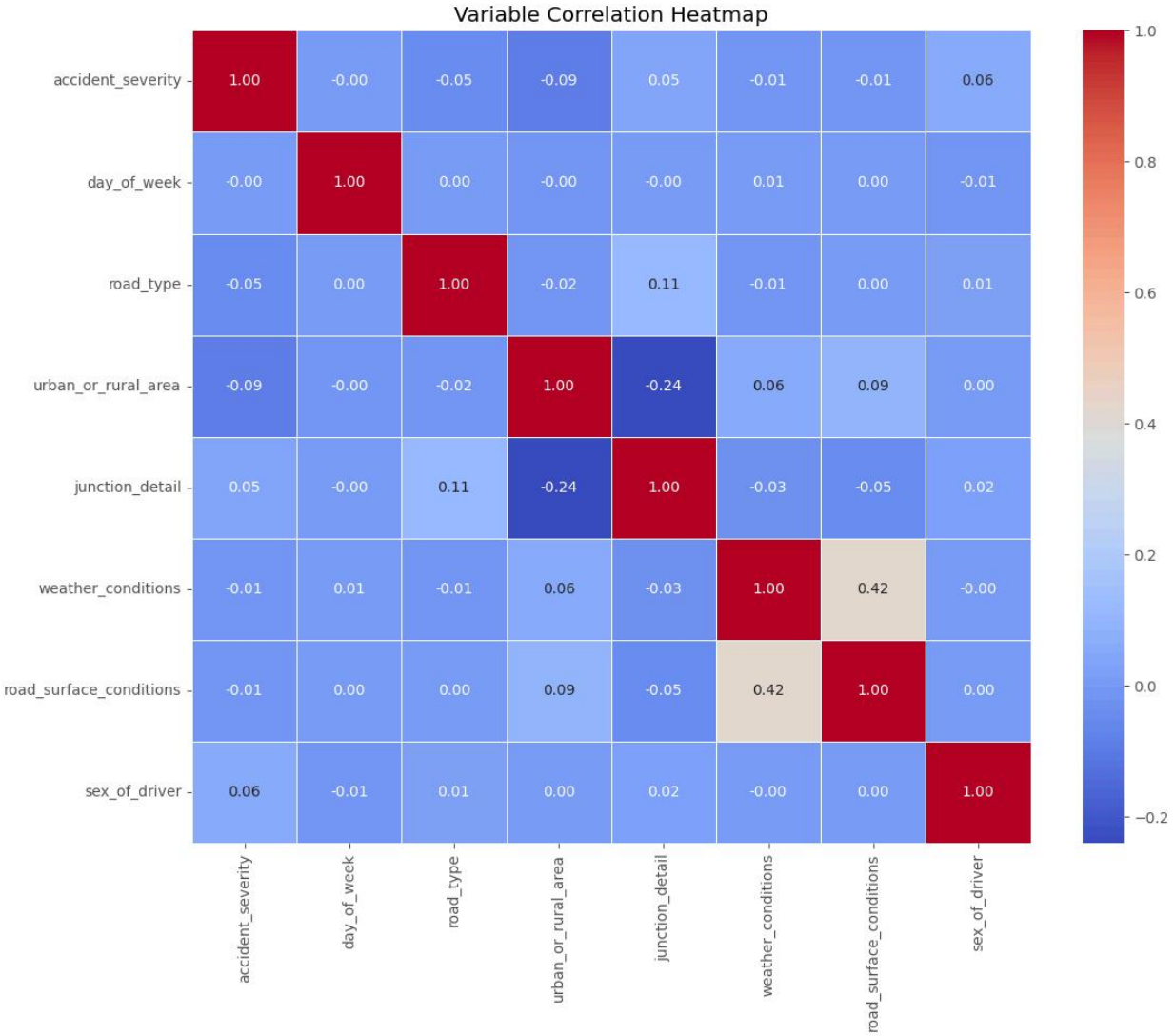

Relationship between road_surface_conditions and Accident Severity
Fatal
Serious
Slight
Number of Accidents
0
5000
10000
15000
20000
25000
30000
35000
1
2
3
4
5
road_surface_conditions


Relationship between road_type and Accident Severity
Fatal
Serious
Slight
Number of Accidents
0
5000
10000
15000
20000
25000
30000
35000
1
2
3
6
7
road_type


Relationship between sex_of_driver and Accident Severity
Fatal
Serious
Slight
Number of Accidents
0
5000
10000
15000
20000
25000
30000
35000
1
2
sex_of_driver


Relationship between urban_or_rural_area and Accident Severity
Fatal
Serious
Slight
Number of Accidents
0
5000
10000
15000
20000
25000
30000
35000
1
2
urban_or_rural_area


Relationship between weather_conditions and Accident Severity
Fatal
Serious
Slight
Number of Accidents
0
10000
20000
30000
40000
1
2
3
4
5
6
7
weather_conditions



The data shows clear class imbalance in accident severity. Weekends, non-junctions, wet surfaces, single carriageways, male drivers, and fog conditions all show higher proportions of fatal accidents despite lower frequencies. Urban areas have more accidents overall. Correlation analysis reveals weak relationships between individual factors and severity, with no significant multicollinearity issues among variables.

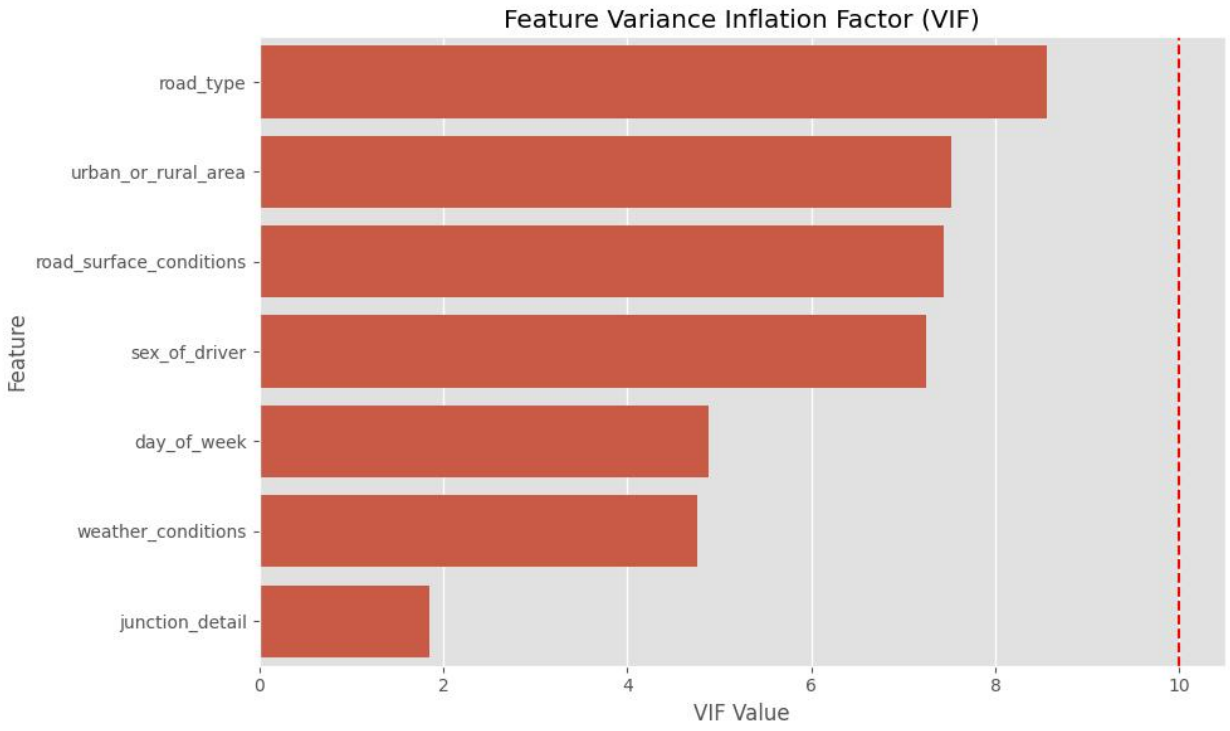
5.2 Feature Selection and Importance Analysis

In [11]:
```
# Feature Selection and Importance Analysis
from sklearn.feature_selection import SelectKBest, chi2, f_classif
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv('https://raw.githubusercontent.com/EvanFeng11/CASA06_Assessment/main/data/cleaned_accident_data.csv')

# Make a copy to avoid modifying the original data
df_encoded = df.copy()

# Identify categorical features
categorical_features = df.select_dtypes(include=['object']).columns.tolist()
if 'severity_category' in categorical_features:
    categorical_features.remove('severity_category')

# Use LabelEncoder for categorical features
for col in categorical_features:
    # First convert to string to ensure uniform type
    df_encoded[col] = df_encoded[col].astype(str)
    # Then apply LabelEncoder
    le = LabelEncoder()
    df_encoded[col] = le.fit_transform(df_encoded[col])

# Sample data to reduce memory usage (optional)
# df_encoded = df_encoded.sample(frac=0.3, random_state=42)
```

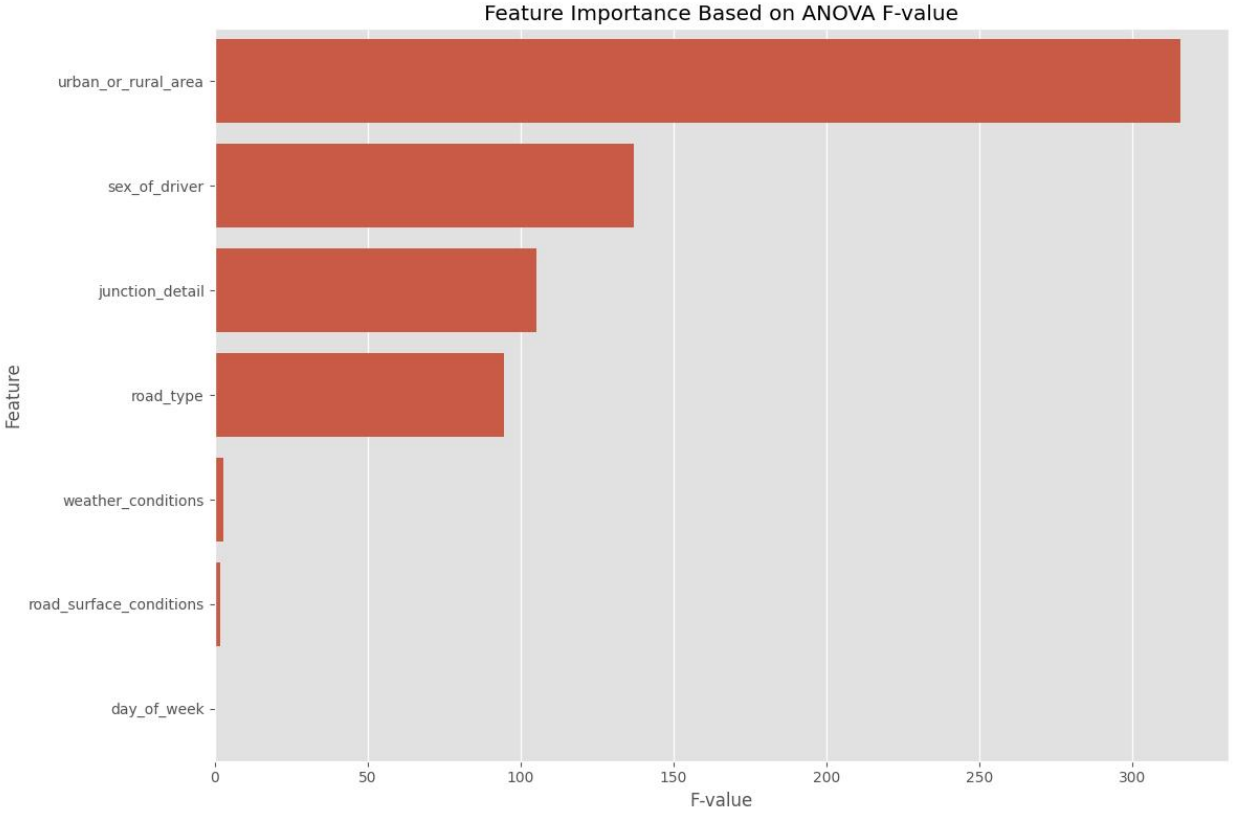
```
# print(f"Using {len(df_encoded)} samples out of {len(df)} total")

# Prepare features and target variable
X = df_encoded.drop(['accident_index', 'accident_severity', 'severity_category'], axis=1, errors='ignore')
y = df_encoded['accident_severity']

# Check data types and dimensions
print("Feature data types:")
print(X.dtypes.value_counts())
print(f"X shape: {X.shape}")
print("\nFirst few rows of processed features:")
print(X.head())

# Ensure all data is numeric
X = X.apply(pd.to_numeric, errors='coerce')
X = X.fillna(0)  # Replace NaN values with 0

# Multicollinearity detection - limit to first 10 features to avoid memory issues
def calculate_vif(X):
    vif_data = pd.DataFrame()
    vif_data["feature"] = X.columns
    vif_data["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    return vif_data.sort_values('VIF', ascending=False)

# Calculate VIF for a limited number of features
if X.shape[1] > 1:  # Need at least two features
    # Limit to 10 features for VIF calculation
    features_for_vif = X.columns[:10]
    print(f"\nCalculating VIF for {len(features_for_vif)} features only")

    vif_df = calculate_vif(X[features_for_vif])
    print("\nMulticollinearity Detection (VIF):")
    print(vif_df)

    # Visualize VIF values
    plt.figure(figsize=(10, 6))
    sns.barplot(x='VIF', y='feature', data=vif_df)
    plt.title('Feature Variance Inflation Factor (VIF)')
    plt.xlabel('VIF Value')
    plt.ylabel('Feature')
    plt.axvline(x=10, color='r', linestyle='--')  # VIF=10 is often considered as threshold
    plt.tight_layout()
    plt.show()

# Use f_classif for feature selection (doesn't require non-negative values)
print("\nUsing f_classif for feature selection:")
k_features = min(15, X.shape[1])  # Limit to 15 features
f_selector = SelectKBest(f_classif, k=k_features)
X_f = f_selector.fit_transform(X, y)

# Get selected features
f_support = f_selector.get_support()
f_features = X.columns[f_support]
f_scores = pd.DataFrame({
    'feature': X.columns[f_support],
    'score': f_selector.scores_[f_support]
}).sort_values('score', ascending=False)

# Visualize f_classif results
plt.figure(figsize=(12, 8))
sns.barplot(x='score', y='feature', data=f_scores)
plt.title('Feature Importance Based on ANOVA F-value')
plt.xlabel('F-value')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()

print("\nTop features selected by ANOVA F-test:")
print(f_scores)

# Try using chi2 with non-negative data
# Make sure all values are non-negative
X_non_neg = X.copy()
X_min = X_non_neg.min()
if X_min.min() < 0:
    print(f"Shifting data to make all values non-negative (minimum value was {X_min.min()})")
    X_non_neg = X_non_neg - X_min + 0.1  # Add small constant to avoid zeros

try:
    print("\nUsing Chi-Square test for feature selection:")
    chi2_selector = SelectKBest(chi2, k=k_features)
    X_chi2 = chi2_selector.fit_transform(X_non_neg, y)

    # Get selected features
    chi2_support = chi2_selector.get_support()
    chi2_features = X.columns[chi2_support]
```

```
    chi2_scores = pd.DataFrame({
        'feature': X.columns[chi2_support],
        'score': chi2_selector.scores_[chi2_support]
    }).sort_values('score', ascending=False)

    # Visualize chi-square results
    plt.figure(figsize=(12, 8))
    sns.barplot(x='score', y='feature', data=chi2_scores)
    plt.title('Feature Importance Based on Chi-Square Test')
    plt.xlabel('Chi-Square Statistic')
    plt.ylabel('Feature')
    plt.tight_layout()
    plt.show()

    print("\nTop features selected by Chi-Square test:")
    print(chi2_scores)

    # Compare feature selection results
    common_features = set(f_features).intersection(set(chi2_features))
    print(f"\nNumber of common features selected by both methods: {len(common_features)}")
    print(f"Common features: {common_features}")
except Exception as e:
    print(f"Chi-Square test failed: {e}")
    print("Continuing with F-test results only.")
```

```
Feature data types:
int64    7
Name: count, dtype: int64
X shape: (68945, 7)

First few rows of processed features:
   day_of_week  road_type  urban_or_rural_area  junction_detail  \
0            1          6                    1                3
1            1          1                    1                1
2            1          6                    1                8
3            1          6                    1                6
4            1          3                    2                0

   weather_conditions  road_surface_conditions  sex_of_driver
0                   1                        1              1
1                   1                        1              1
2                   1                        1              1
3                   1                        2              1
4                   1                        1              1

Calculating VIF for 7 features only

Multicollinearity Detection (VIF):
                  feature  VIF
1               road_type 8.56
2     urban_or_rural_area 7.53
5  road_surface_conditions 7.44
6           sex_of_driver 7.25
0             day_of_week 4.88
4      weather_conditions 4.77
3         junction_detail 1.85
```



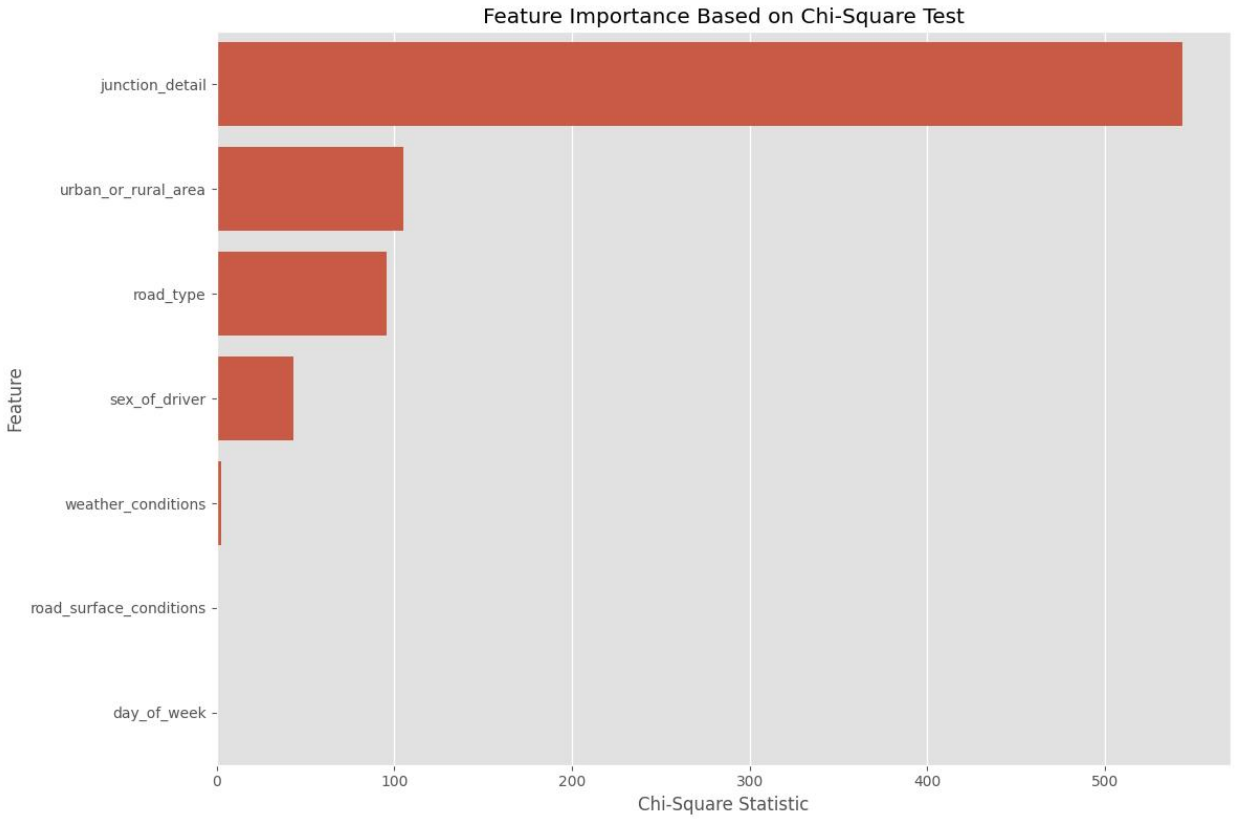
Using f_classif for feature selection:



Top features selected by ANOVA F-test:

	feature	score
2	urban_or_rural_area	315.73
6	sex_of_driver	137.13
3	junction_detail	105.19
1	road_type	94.48
4	weather_conditions	2.97
5	road_surface_conditions	1.92
0	day_of_week	0.26

Using Chi-Square test for feature selection:




```
Top features selected by Chi-Square test:
                   feature  score
3          junction_detail 543.46
2      urban_or_rural_area 104.90
1                road_type  95.95
6             sex_of_driver  43.47
4       weather_conditions   2.48
5  road_surface_conditions   0.93
0              day_of_week   0.46

Number of common features selected by both methods: 7
Common features: {'road_type', 'sex_of_driver', 'urban_or_rural_area', 'junction_detail', 'day_of_week', 'road_surface_cond
itions', 'weather_conditions'}
```

We employed three complementary methods to identify key factors affecting accident severity: VIF analysis for multicollinearity, ANOVA F-test, and Chi-Square test for feature importance.

VIF analysis showed all variables below the critical threshold of 10, with road type (8.56), urban/rural area (7.53), and road surface conditions (7.44) having the highest values, while junction detail (1.85) had the lowest.

ANOVA F-test identified urban/rural area as most important (F=315.73), followed by driver sex (137.13), junction detail (105.19), and road type (94.48). Weather conditions, road surface conditions, and day of week showed weaker relationships.

Chi-Square test ranked junction detail highest (543.46), followed by urban/rural area (104.90) and road type (95.95).

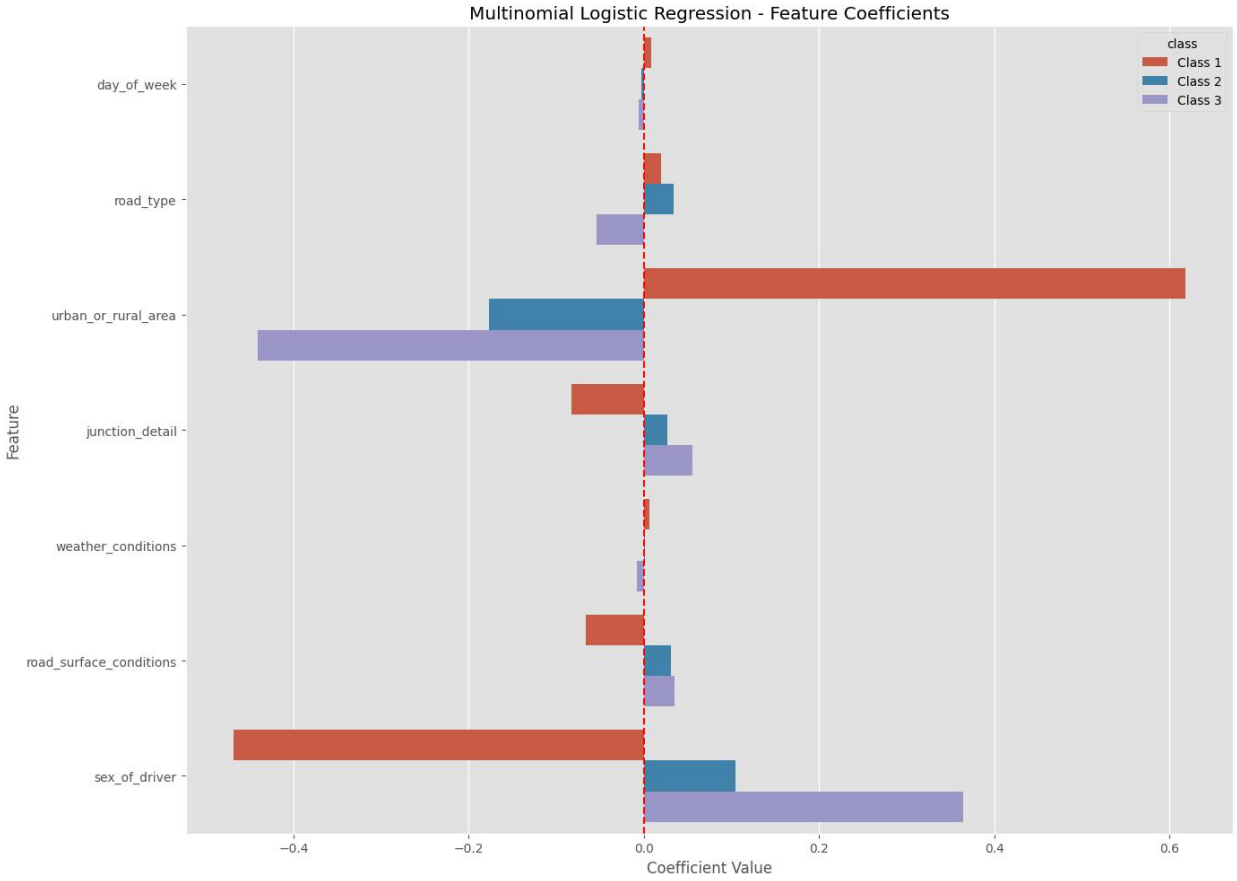
All seven features were retained for modeling, with emphasis on urban/rural area, junction detail, road type, and driver sex as most influential factors.

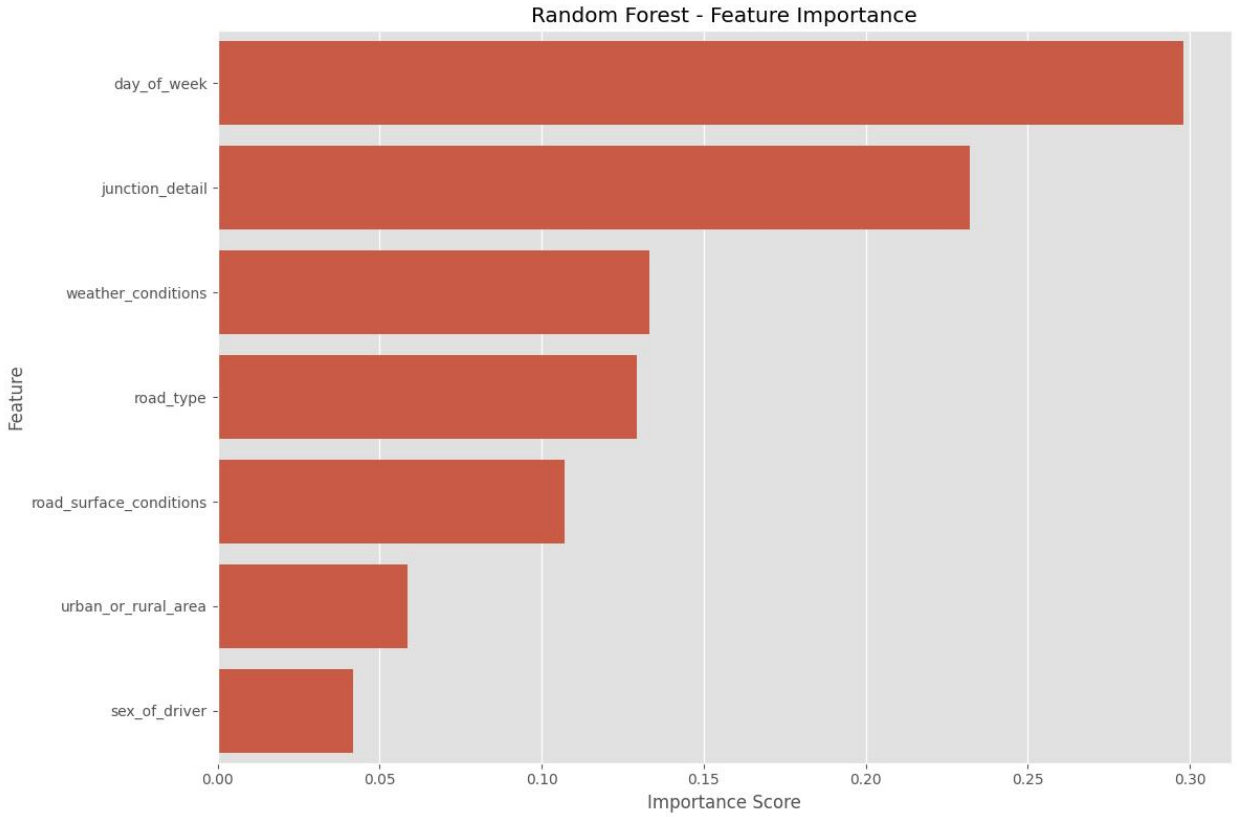
5.3 Predictive Model Development

In [12]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import statsmodels.formula.api as smf

# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Ordinal Logistic Regression
# Prepare formula for statsmodels
formula_parts = []
for col in X.columns:
    formula_parts.append(f"{col}")

formula = f"accident_severity ~ {' + '.join(formula_parts)}"

# Train ordinal logistic regression model
try:
    ordinal_model = smf.ologit(formula, data=df_encoded).fit()
```

```
    print(ordinal_model.summary())

    # Extract and visualize coefficients
    coef_df = pd.DataFrame({
        'feature': ordinal_model.params.index[1:],  # Skip intercept
        'coefficient': ordinal_model.params.values[1:],
        'p_value': ordinal_model.pvalues.values[1:]
    })
    coef_df = coef_df.sort_values('coefficient', ascending=False)

    # Visualize significant coefficients
    significant_coef = coef_df[coef_df['p_value'] < 0.05]
    plt.figure(figsize=(12, 8))
    sns.barplot(x='coefficient', y='feature', data=significant_coef.head(15))
    plt.title('Ordinal Logistic Regression - Significant Feature Coefficients')
    plt.xlabel('Coefficient Value')
    plt.ylabel('Feature')
    plt.axvline(x=0, color='r', linestyle='--')
    plt.tight_layout()
    plt.show()
except Exception as e:
    print(f"Ordinal Logistic Regression model training failed: {e}")
    print("Using alternative method...")
    # If ordinal logistic regression fails, use multinomial logistic regression as alternative
    from sklearn.linear_model import LogisticRegression
    lr = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)
    lr.fit(X_train, y_train)

    # Extract coefficients
    coef_df = pd.DataFrame()
    for i, target in enumerate(lr.classes_):
        temp_df = pd.DataFrame({
            'feature': X.columns,
            'coefficient': lr.coef_[i],
            'class': f'Class {target}'
        })
        coef_df = pd.concat([coef_df, temp_df])

    # Visualize coefficients
    plt.figure(figsize=(14, 10))
    sns.barplot(x='coefficient', y='feature', hue='class', data=coef_df.head(45))
    plt.title('Multinomial Logistic Regression - Feature Coefficients')
    plt.xlabel('Coefficient Value')
    plt.ylabel('Feature')
    plt.axvline(x=0, color='r', linestyle='--')
    plt.tight_layout()
    plt.show()

# Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)

# Feature importance
feature_importances = pd.DataFrame({
    'feature': X.columns,
    'importance': rf.feature_importances_
}).sort_values('importance', ascending=False)

# Visualize feature importance
plt.figure(figsize=(12, 8))
sns.barplot(x='importance', y='feature', data=feature_importances.head(15))
plt.title('Random Forest - Feature Importance')
plt.xlabel('Importance Score')
plt.ylabel('Feature')
plt.tight_layout()
plt.show()

print("Top 15 features by Random Forest importance:")
print(feature_importances.head(15))
```

```
Ordinal Logistic Regression model training failed: module 'statsmodels.formula.api' has no attribute 'ologit'
Using alternative method...
```






```
Top 15 features by Random Forest importance:
                   feature  importance
0             day_of_week        0.30
3          junction_detail        0.23
4       weather_conditions        0.13
1               road_type        0.13
5  road_surface_conditions        0.11
2      urban_or_rural_area        0.06
6            sex_of_driver        0.04
```

First, we implement Ordinal Logistic Regression because accident severity is an ordered categorical variable (Fatal > Serious > Slight). This model respects the ordinal nature of our target variable and provides interpretable coefficients. We include a

fallback to multinomial logistic regression in case the ordinal model fails to converge. Second, we train a Random Forest Classifier.

5.3.1 Multinomial Logistic Regression Results

Key coefficient insights:

- **Urban/rural area** showed strongest influence (±0.6 coefficient), with rural areas strongly associated with fatal accidents
- **Driver sex** demonstrated notable impact, with male drivers associated with higher fatal accident probability
- **Junction detail** and **road surface conditions** showed moderate influence
- **Weather conditions** and **day of week** had minimal direct influence

5.3.2 Random Forest Model Analysis

Feature importance ranking:

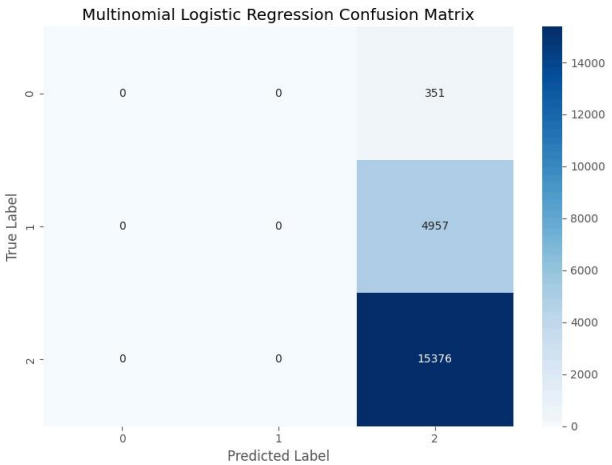
- **Day of week** ranked highest (0.30), suggesting complex non-linear relationships
- **Junction detail** ranked second (0.23)
- **Weather conditions** and **road type** (0.13 each)
- **Urban/rural area** (0.06) and **driver sex** (0.04) ranked lower than in statistical tests

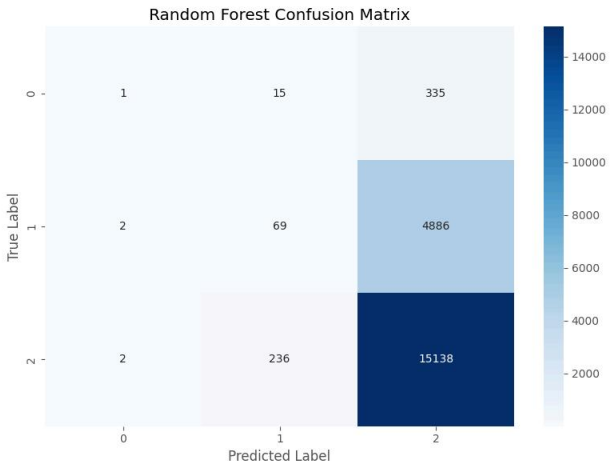
5.4 Model Evaluation and Comparison

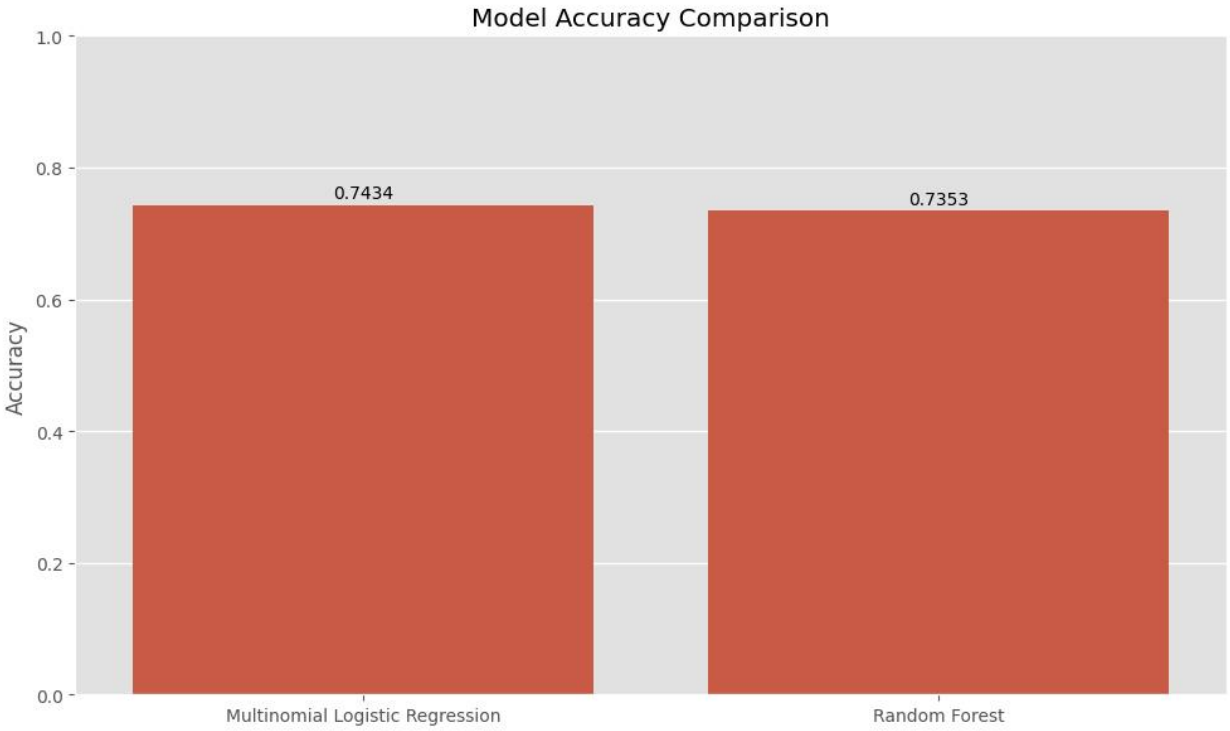
In [13]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score

# Prepare predictions and actual values
# Ordinal logistic regression predictions
try:
    ordinal_pred_probs = ordinal_model.predict(X_test)
    ordinal_pred = np.argmax(ordinal_pred_probs, axis=1) + 1  # +1 because classes start from 1
    has_ordinal = True
except:
    # If using multinomial logistic regression
    lr_pred = lr.predict(X_test)
    has_ordinal = False

# Random forest predictions
rf_pred = rf.predict(X_test)

# Calculate accuracy
if has_ordinal:
    ordinal_accuracy = accuracy_score(y_test, ordinal_pred)
    print(f"Ordinal Logistic Regression accuracy: {ordinal_accuracy:.4f}")
else:
    lr_accuracy = accuracy_score(y_test, lr_pred)
    print(f"Multinomial Logistic Regression accuracy: {lr_accuracy:.4f}")

rf_accuracy = accuracy_score(y_test, rf_pred)
print(f"Random Forest accuracy: {rf_accuracy:.4f}")

# Classification reports
if has_ordinal:
    print("\nOrdinal Logistic Regression Classification Report:")
    print(classification_report(y_test, ordinal_pred))
else:
    print("\nMultinomial Logistic Regression Classification Report:")
    print(classification_report(y_test, lr_pred))

print("\nRandom Forest Classification Report:")
print(classification_report(y_test, rf_pred))

# Confusion matrix visualization
fig, axes = plt.subplots(1, 2, figsize=(16, 6))

# Logistic regression confusion matrix
if has_ordinal:
    sns.heatmap(confusion_matrix(y_test, ordinal_pred), annot=True, fmt='d', cmap='Blues', ax=axes[0])
    axes[0].set_title('Ordinal Logistic Regression Confusion Matrix')
else:
    sns.heatmap(confusion_matrix(y_test, lr_pred), annot=True, fmt='d', cmap='Blues', ax=axes[0])
    axes[0].set_title('Multinomial Logistic Regression Confusion Matrix')
axes[0].set_xlabel('Predicted Label')
axes[0].set_ylabel('True Label')

# Random forest confusion matrix
sns.heatmap(confusion_matrix(y_test, rf_pred), annot=True, fmt='d', cmap='Blues', ax=axes[1])
axes[1].set_title('Random Forest Confusion Matrix')
axes[1].set_xlabel('Predicted Label')
axes[1].set_ylabel('True Label')
```

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

# Model performance comparison chart
if has_ordinal:
    models = ['Ordinal Logistic Regression', 'Random Forest']
    accuracies = [ordinal_accuracy, rf_accuracy]
else:
    models = ['Multinomial Logistic Regression', 'Random Forest']
    accuracies = [lr_accuracy, rf_accuracy]

plt.figure(figsize=(10, 6))
sns.barplot(x=models, y=accuracies)
plt.title('Model Accuracy Comparison')
plt.ylabel('Accuracy')
plt.ylim(0, 1)
for i, acc in enumerate(accuracies):
    plt.text(i, acc + 0.01, f'{acc:.4f}', ha='center')
plt.tight_layout()
plt.show()

# Cross-validation (Random Forest)
cv_scores = cross_val_score(rf, X, y, cv=5)
print(f"\nRandom Forest 5-fold cross-validation accuracy: {cv_scores.mean():.4f} (±{cv_scores.std():.4f})")
```

```
Multinomial Logistic Regression accuracy: 0.7434
Random Forest accuracy: 0.7353

Multinomial Logistic Regression Classification Report:
              precision    recall  f1-score   support

           1       0.00      0.00      0.00       351
           2       0.00      0.00      0.00      4957
           3       0.74      1.00      0.85     15376

    accuracy                           0.74     20684
   macro avg       0.25      0.33      0.28     20684
weighted avg       0.55      0.74      0.63     20684


Random Forest Classification Report:
              precision    recall  f1-score   support

           1       0.20      0.00      0.01       351
           2       0.22      0.01      0.03      4957
           3       0.74      0.98      0.85     15376

    accuracy                           0.74     20684
   macro avg       0.39      0.33      0.29     20684
weighted avg       0.61      0.74      0.64     20684
```








```
Random Forest 5-fold cross-validation accuracy: 0.7363 (±0.0012)
```

Both models demonstrated similar overall accuracy: Multinomial Logistic Regression achieved 74.34%, while Random Forest reached 73.53%. Random Forest showed good stability through 5-fold cross-validation with 73.63%(±0.12%) average accuracy.

Classification report analysis revealed significant challenges with class imbalance:

- Logistic Regression showed F1-scores of 0.00 for both fatal and serious accidents, indicating complete failure to identify minority classes
- Random Forest achieved F1-scores of 0.01 for fatal and 0.03 for serious accidents, slightly better but still limited
- Both models performed similarly for slight accidents (F1≈0.85)

Confusion matrix analysis showed:

- Logistic Regression predicted all samples as slight accidents
- Random Forest correctly identified a small number of minority instances (1 fatal and 69 serious accidents)

Overall, while both models performed reasonably in terms of accuracy, they demonstrated significant limitations in identifying high-risk minority class accidents, highlighting the need for further optimization to address class imbalance.

6 Conclusion

[go back to the top]

Our comprehensive analysis of London road accidents addressed our primary research questions regarding factors influencing accident severity and the effectiveness of predictive models. We found that geographical context plays a crucial role, with rural areas strongly associated with fatal accidents despite London being predominantly urban. Driver characteristics significantly impact outcomes, with male drivers linked to higher fatality rates across statistical tests and regression models. Road infrastructure factors, particularly junction complexity and road type (motorways and A-roads), consistently emerged as strong predictors of severity. Environmental conditions, including road surface and weather, showed moderate influence, while day of week demonstrated complex non-linear relationships with severity in our Random Forest model.

Regarding predictive capabilities, both Multinomial Logistic Regression and Random Forest models achieved similar overall accuracy (~74%), demonstrating reasonable performance. However, both struggled significantly with the severe class imbalance, showing limited ability to identify the minority classes of fatal and serious accidents. The Random Forest model demonstrated slightly better discrimination, correctly identifying a small number of fatal and serious accidents, while the logistic regression model classified virtually all instances as slight accidents. These findings highlight that while machine learning approaches provide valuable predictive insights for road safety management, significant challenges remain in accurately identifying high-risk, low-frequency events.

7 References

[go back to the top]

AlHashmi, M. Y. (2024). Using Machine Learning for Road Accident Severity Prediction and Optimal Rescue Pathways (Order No. 31767419). Available from ProQuest Dissertations & Theses Global. (3148720460). https://www.proquest.com/dissertations-theses/using-machine-learning-road-accident-severity/docview/3148720460/se-2

Behboudi, N., Moosavi, S., & Ramnath, R. (2024). Recent Advances in Traffic Accident Analysis and Prediction: A Comprehensive Review of Machine Learning Techniques. https://doi.org/10.48550/arxiv.2406.13968

Michalaki, P., Quddus, M. A., Pitfield, D., & Huetson, A. (2015). Exploring the factors affecting motorway accident severity in England using the generalised ordered logistic regression model. Journal of Safety Research, 55, 89–97. https://doi.org/10.1016/j.jsr.2015.09.004

Wang, C., Quddus, M. A., & Ison, S. G. (2013). The effect of traffic and road characteristics on road safety: A review and future research direction. Safety Science, 57, 264–275. https://doi.org/10.1016/j.ssci.2013.02.012