Report: Analysis of Road Traffic Accidents in the UK between the years 2009,2015 and 2016.

A Study on the Correlation between Vehicle Type and Casualty Severity in the UK road accidents.

1. **Introduction:**

Road accidents represent a critical societal challenge with profound implications for public safety and welfare. As a pervasive global issue, the analysis of road accident data plays a crucial role in understanding the underlying factors contributing to incidents and formulating effective preventative strategies. This project focuses on investigating road accidents in the United Kingdom during the years 2009, 2015, and 2016.

The primary objective of this research is to explore and analyze the characteristics of road accidents, particularly focusing on casualty severity, vehicle types, and associated factors. By examining data spanning multiple years, we aim to discern patterns, trends, and key insights that can inform evidence-based policy decisions and improve road safety measures.
The selected timeframe, encompassing the years 2009, 2015, and 2016, provides a comprehensive snapshot of road accidents in the UK. These years are strategically chosen to capture both historical trends and contemporary challenges, facilitating a nuanced understanding of the evolving dynamics of road safety.

2. **Methods**

2.1 Data source

In this project there were three data sets, which has been considered into analysis. Datasets used for this analysis is sourced from Data Mill North and European Data Portal website, a

platform that provides access to a wide range of public datasets. The road accident data is a comprehensive collection that includes information on variables such as accident location, time, vehicle types, and casualty details. Utilizing this dataset ensures the richness and reliability of the information used in our analysis.
As there are three similar datasets, I explain about one of them.

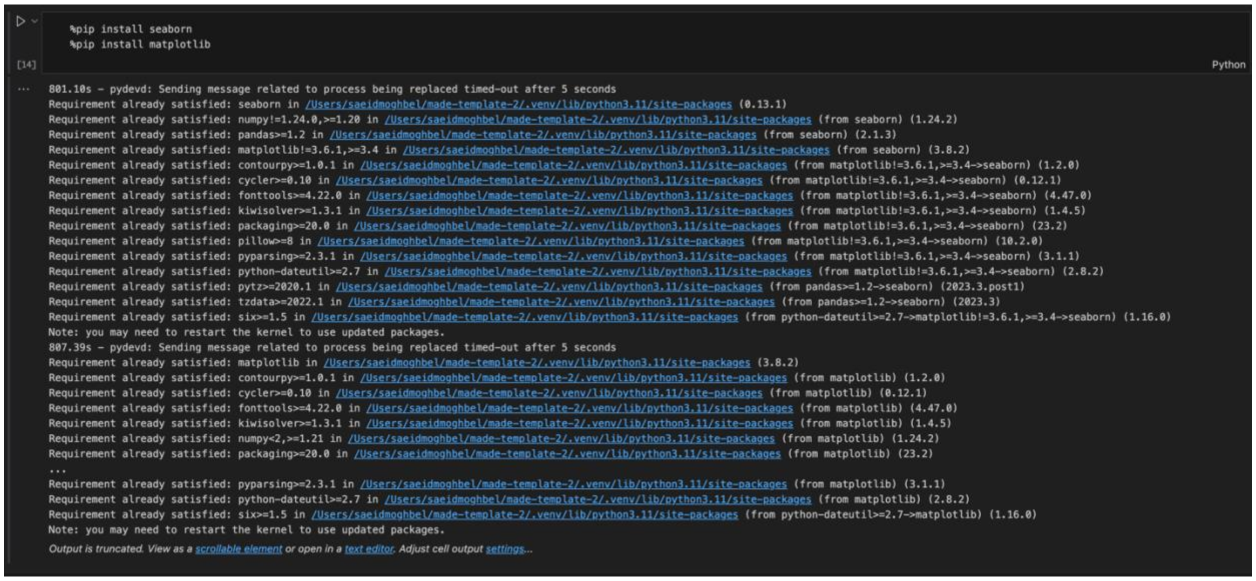
Data set: The UK Road accidents in the year 2009

- URL: UK road accidents in 2009
- The data set provided information about the road accidents in the UK for the year 2009. It also includes information about the date and hour of the accident, also the type of vehicles, age and sex of casualties and also casualty severity.

Columns:
Easting and Northing: Geographical information about the place of occurrence of accident.
Number of Vehicle: Numerical information about how many vehicles were in the accident.
Accident Date: Including the dd-mm-yyyy date format to show the date of the accident.
Time: including the time of occurring the accident.
Road surface: About the state of the road, including (Dry, Wet/Damp).
Weather condition: About the weather situation in that time.
Casualty Class: Whether the casualties are Pedestrian, Driver or Passenger.
Casualty Severity: The severity of casualty.
Sex of Casualty: Whether the casualties were men or women.
Age of Casualty: How old were the casualties.
Type of Vehicle: What is the type of vehicles in accident.

2.2 Install Dependencies

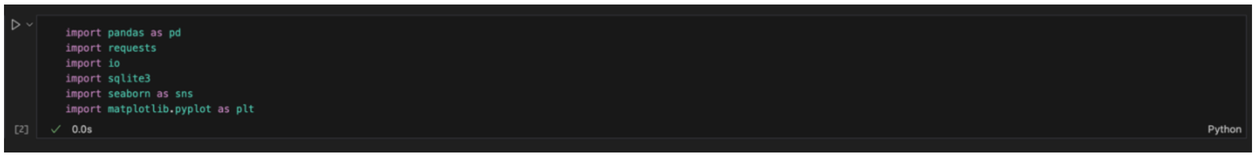
In this data analysis project, SQLite3 has been used to load data and store it, but it did not need to install separately, as it is pre-installed in python3.
Pandas also used as it is designed for data manipulation and analysis, it also provides powerful data structures (DataFrame) that simplify working with structured data.
Requests used for making HTTP request in python, and it is ideal for fetching data from web resources.
Seaborn and matplotlib are two other libraries were used for representing the plot information. Seaborn is well-suited for creating count plots to visualize the distribution of casualty severity across different vehicle types. Matplotlib is also Used in conjunction with seaborn for displaying count plots and other visualizations.



```
%pip install seaborn
%pip install matplotlib
```

```
801.10s - pydevd: Sending message related to process being replaced timed-out after 5 seconds
Requirement already satisfied: seaborn in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (0.13.1)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from seaborn) (1.24.2)
Requirement already satisfied: pandas>=1.2 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from seaborn) (2.1.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from seaborn) (3.8.2)
Requirement already satisfied: contourpy>=1.0.1 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (4.47.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (23.2)
Requirement already satisfied: pillow>=8 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from pandas>=1.2->seaborn) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from pandas>=1.2->seaborn) (2023.3)
Requirement already satisfied: six>=1.5 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
807.39s - pydevd: Sending message related to process being replaced timed-out after 5 seconds
Requirement already satisfied: matplotlib in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (3.8.2)
Requirement already satisfied: contourpy>=1.0.1 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib) (4.47.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib) (1.4.5)
Requirement already satisfied: numpy<2,>=1.21 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib) (1.24.2)
Requirement already satisfied: packaging>=20.0 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib) (23.2)
...
Requirement already satisfied: pyparsing>=2.3.1 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib) (3.1.1)
Requirement already satisfied: python-dateutil>=2.7 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /Users/saeidmoghbel/made-template-2/.venv/lib/python3.11/site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

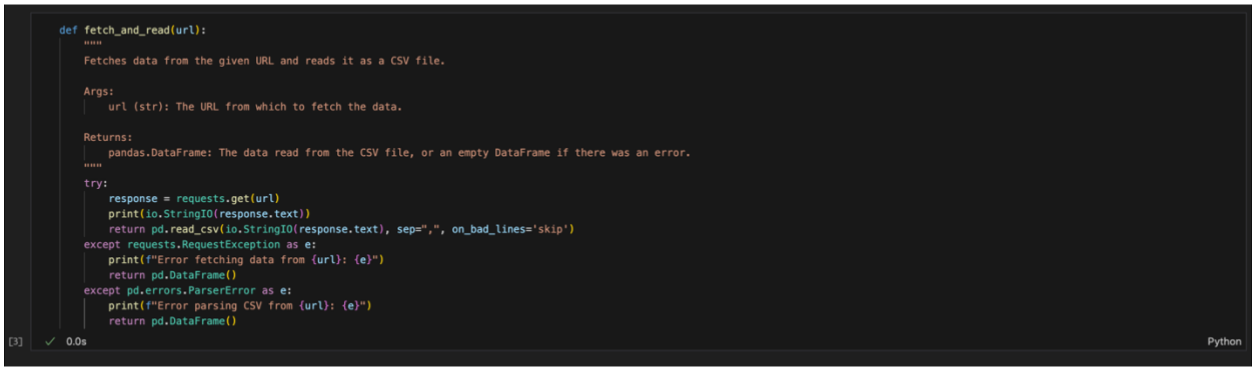
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...



```
import pandas as pd
import requests
import io
import sqlite3
import seaborn as sns
import matplotlib.pyplot as plt
```

2.3 Data Pipeline and Data Cleaning

In the implemented data pipeline for analyzing UK road accident data, the process begins with the retrieval of information from external sources, specifically URLs hosting the road accident datasets. The **fetch_and_read** function utilizes the **requests** library to fetch data through HTTP requests, and the response text is seamlessly handled using the **io** module to facilitate reading into **pandas** DataFrames.

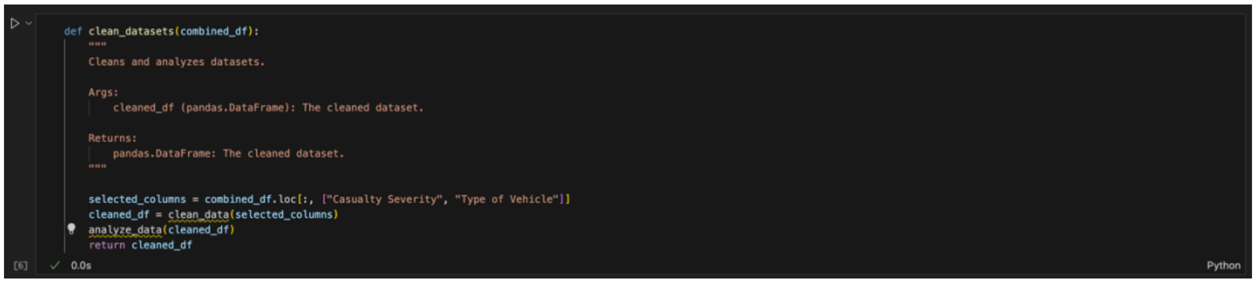


```python
def fetch_and_read(url):
    """
    Fetches data from the given URL and reads it as a CSV file.

    Args:
        url (str): The URL from which to fetch the data.

    Returns:
        pandas.DataFrame: The data read from the CSV file, or an empty DataFrame if there was an error.
    """
    try:
        response = requests.get(url)
        print(io.StringIO(response.text))
        return pd.read_csv(io.StringIO(response.text), sep=",", on_bad_lines='skip')
    except requests.RequestException as e:
        print(f"Error fetching data from {url}: {e}")
        return pd.DataFrame()
    except pd.errors.ParserError as e:
        print(f"Error parsing CSV from {url}: {e}")
        return pd.DataFrame()
```
```
[3]  ✓ 0.0s                                                    Python
```

Following data retrieval, the script transitions into a crucial phase of loading and preprocessing. Raw data is loaded into DataFrames using the powerful **pandas** library, and the **clean_data** function undertakes preprocessing tasks, such as handling missing values and standardizing string formats, ensuring the data is structured for meaningful analysis.

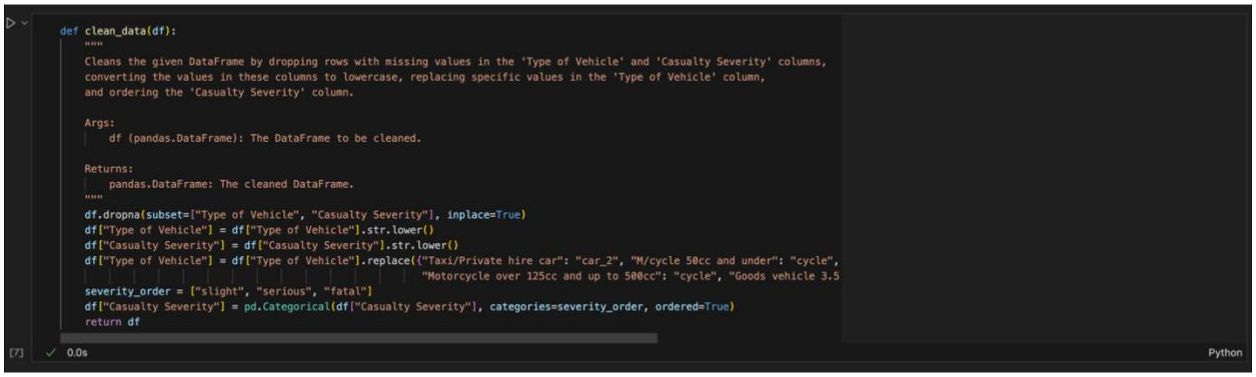


```python
def clean_datasets(combined_df):
    """
    Cleans and analyzes datasets.

    Args:
        cleaned_df (pandas.DataFrame): The cleaned dataset.

    Returns:
        pandas.DataFrame: The cleaned dataset.
    """

    selected_columns = combined_df.loc[:, ["Casualty Severity", "Type of Vehicle"]]
    cleaned_df = clean_data(selected_columns)
    analyze_data(cleaned_df)
    return cleaned_df
```
```
[6]  ✓ 0.0s                                                    Python
```

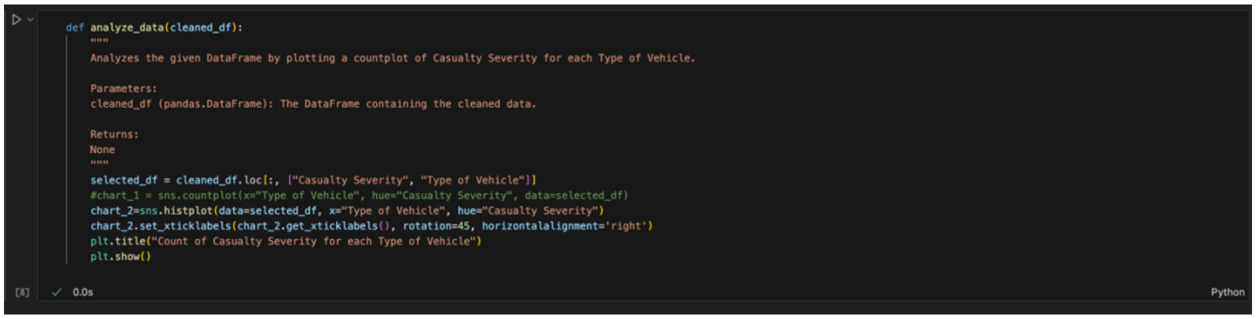


```python
def clean_data(df):
    """
    Cleans the given DataFrame by dropping rows with missing values in the 'Type of Vehicle' and 'Casualty Severity' columns,
    converting the values in these columns to lowercase, replacing specific values in the 'Type of Vehicle' column,
    and ordering the 'Casualty Severity' column.

    Args:
        df (pandas.DataFrame): The DataFrame to be cleaned.

    Returns:
        pandas.DataFrame: The cleaned DataFrame.
    """
    df.dropna(subset=["Type of Vehicle", "Casualty Severity"], inplace=True)
    df["Type of Vehicle"] = df["Type of Vehicle"].str.lower()
    df["Casualty Severity"] = df["Casualty Severity"].str.lower()
    df["Type of Vehicle"] = df["Type of Vehicle"].replace({"Taxi/Private hire car": "car_2", "M/cycle 50cc and under": "cycle",
                                                          "Motorcycle over 125cc and up to 500cc": "cycle", "Goods vehicle 3.5
    severity_order = ["slight", "serious", "fatal"]
    df["Casualty Severity"] = pd.Categorical(df["Casualty Severity"], categories=severity_order, ordered=True)
    return df
```
```
[7]  ✓ 0.0s                                                    Python
```

The subsequent step involves exploratory data analysis and visualization, where the **seaborn** and **matplotlib** libraries come into play. The **analyze_data** function generates insightful visualizations, such as countplots, offering a comprehensive understanding of the distribution of casualty severity across different vehicle types.



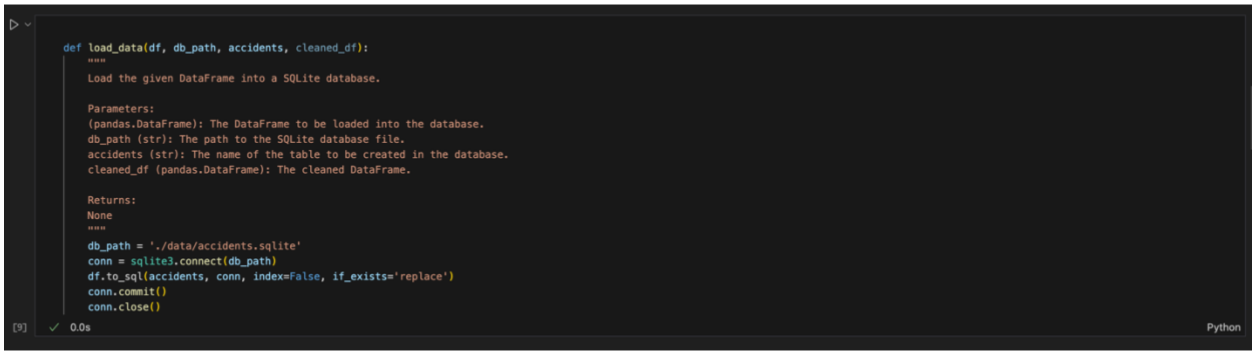
```python
def analyze_data(cleaned_df):
    """
    Analyzes the given DataFrame by plotting a countplot of Casualty Severity for each Type of Vehicle.

    Parameters:
    cleaned_df (pandas.DataFrame): The DataFrame containing the cleaned data.

    Returns:
    None
    """
    selected_df = cleaned_df.loc[:, ["Casualty Severity", "Type of Vehicle"]]
    #chart_1 = sns.countplot(x="Type of Vehicle", hue="Casualty Severity", data=selected_df)
    chart_2=sns.histplot(data=selected_df, x="Type of Vehicle", hue="Casualty Severity")
    chart_2.set_xticklabels(chart_2.get_xticklabels(), rotation=45, horizontalalignment='right')
    plt.title("Count of Casualty Severity for each Type of Vehicle")
    plt.show()
```

```
[8]  ✓ 0.0s                                                                 Python
```

To ensure data persistence and future use, the script incorporates a data storage step. The **sqlite3** module facilitates the creation of an SQLite database, and the **pandas to_sql** method is employed in the **load_data** function to store cleaned DataFrames into the database.



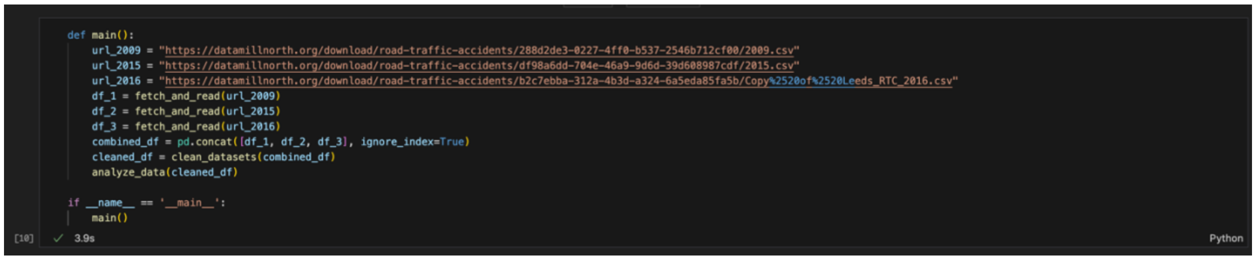
```python
def load_data(df, db_path, accidents, cleaned_df):
    """
    Load the given DataFrame into a SQLite database.

    Parameters:
    (pandas.DataFrame): The DataFrame to be loaded into the database.
    db_path (str): The path to the SQLite database file.
    accidents (str): The name of the table to be created in the database.
    cleaned_df (pandas.DataFrame): The cleaned DataFrame.

    Returns:
    None
    """
    db_path = './data/accidents.sqlite'
    conn = sqlite3.connect(db_path)
    df.to_sql(accidents, conn, index=False, if_exists='replace')
    conn.commit()
    conn.close()
```

```
[9]  ✓ 0.0s                                                                 Python
```

Structural organization and modularity are emphasized in the script, with distinct functions (**fetch_and_read**, **clean_data**, **analyze_data**, **load_data**) encapsulating specific functionalities. This design enhances script readability, maintainability, and reusability. Additionally, the implementation includes error handling mechanisms, employing **try** and **except** blocks to gracefully manage potential errors during data retrieval and processing. Finally, The **main** function serves as the central orchestration point in the data analysis script, encapsulating the core steps of the data pipeline. It initiates the process by fetching road accident data for the years 2009, 2015, and 2016 from designated URLs, utilizing the **fetch_and_read** function to create separate DataFrames for each year. Subsequently, these DataFrames are combined into a unified **combined_df** using the **pd.concat** function. The collected data undergoes a comprehensive cleaning and preprocessing phase through the **clean_datasets** function, ensuring its readiness for analysis. The final cleaned DataFrame is then passed to the **analyze_data** function, generating visualizations to examine the distribution of casualty severity across different types of vehicles. The modular design and logical flow within the **main** function contribute to script readability and maintainability, facilitating a systematic approach to the comprehensive analysis of UK road accident data.



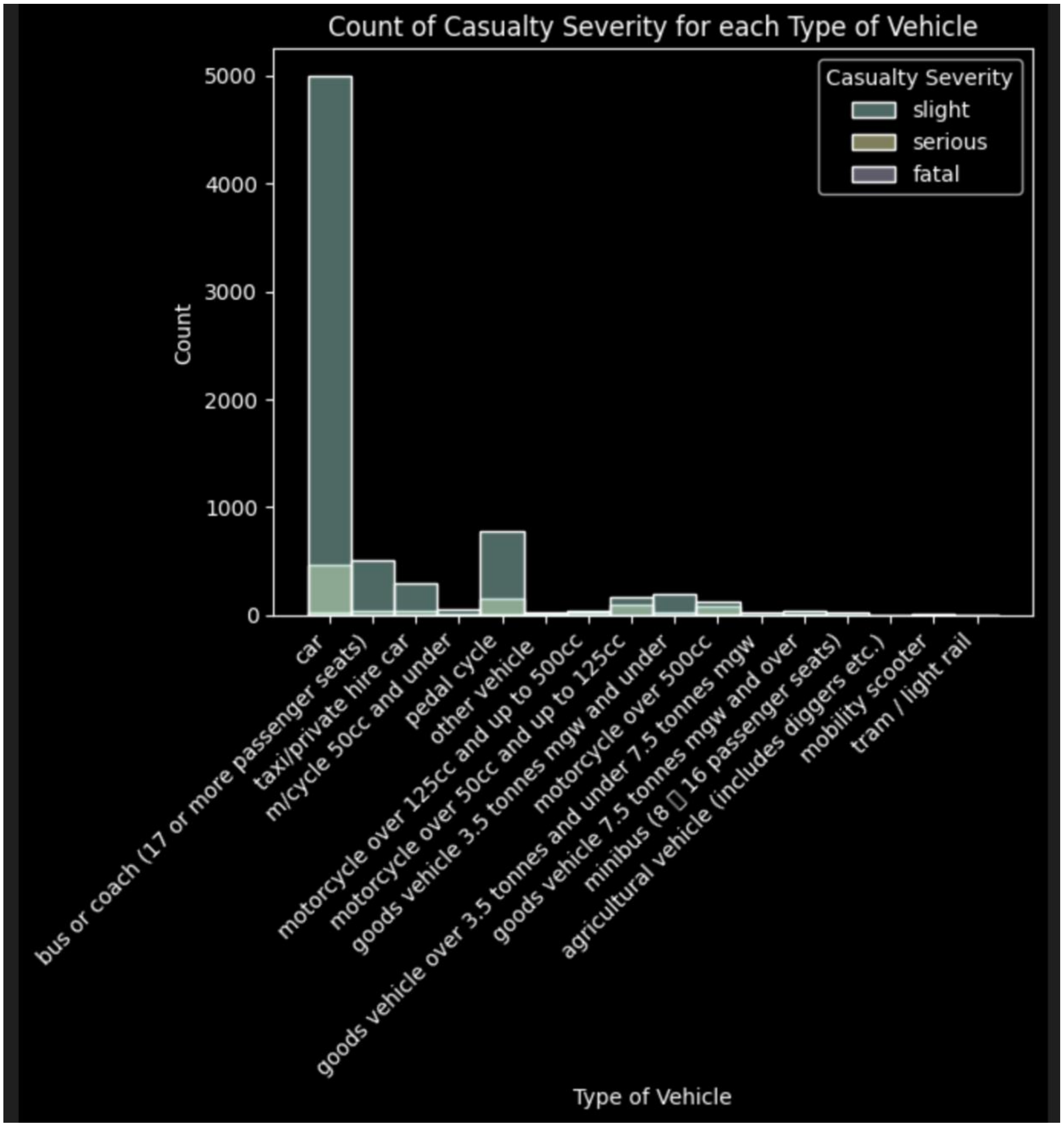
```
def main():
    url_2009 = "https://datamillnorth.org/download/road-traffic-accidents/288d2de3-0227-4ff0-b537-2546b712cf00/2009.csv"
    url_2015 = "https://datamillnorth.org/download/road-traffic-accidents/df98a6dd-704e-46a9-9d6d-39d608987cdf/2015.csv"
    url_2016 = "https://datamillnorth.org/download/road-traffic-accidents/b2c7ebba-312a-4b3d-a324-6a5eda85fa5b/Copy%2520of%2520Leeds_RTC_2016.csv"
    df_1 = fetch_and_read(url_2009)
    df_2 = fetch_and_read(url_2015)
    df_3 = fetch_and_read(url_2016)
    combined_df = pd.concat([df_1, df_2, df_3], ignore_index=True)
    cleaned_df = clean_datasets(combined_df)
    analyze_data(cleaned_df)

if __name__ == '__main__':
    main()
```

[10] ✓ 3.9s Python

3. **Results**

The analysis of the road traffic accidents dataset yielded a comprehensive visualization of casualty severity across different vehicle types, as depicted in the bar chart.
The chart illustrates the count of casualties categorized by severity—slight, serious, and fatal—for each vehicle type.
It is immediately apparent that cars are involved in the majority of accidents with varying sev erity levels, with the count of slight severity being the most predominant.
This is followed by a significantly lower number of incidents involving buses or coaches and motorcycles over 125cc and up to 500cc, which also primarily result in slight casualties.
Interestingly, while the number of accidents involving goods vehicles over 3.5 tonnes and up to 7.5 tonnes is relatively small,
they have a higher proportion of serious casualties compared to other vehicle types.
The data for pedal cycles, motorcycles up to 50cc,
and other vehicle categories show a lower overall number of accidents,
but the distribution across severity categories remains consistent with the trend observed in m ore prevalent vehicle types. The chart also reveals that certain vehicle types,
such as agricultural vehicles and trams/light rail, have a minimal representation in the data, suggesting a lower involvement in road traffic accidents or a possible underreporting in the d ataset.
The stark contrast in the number of accidents associated with cars compared to other ve hicle types raises questions about the factors contributing to this discrepancy and underscores the need for targeted road safety interventions for car drivers and passengers.


Count of Casualty Severity for each Type of Vehicle
Casualty Severity
slight
serious
fatal
Count
5000
4000
3000
2000
1000
0
car
bus or coach (17 or more passenger seats)
taxi/private hire car
m/cycle 50cc and under
pedal cycle
other vehicle
motorcycle over 125cc and up to 500cc
motorcycle over 50cc and up to 125cc
goods vehicle 3.5 tonnes mgw and under
motorcycle over 500cc
goods vehicle over 3.5 tonnes and under 7.5 tonnes mgw
goods vehicle 7.5 tonnes mgw and over
minibus (8 16 passenger seats)
agricultural vehicle (includes diggers etc.)
mobility scooter
tram / light rail
Type of Vehicle