DATA ANALYSIS REPORT

INTRODUCTION:

In contemporary civilizations, one of the biggest threats to public health and safety is traffic accidents. They cause significant financial losses, psychological suffering, and, most tragically, fatalities. Creating strategies and regulations for effective road accident prevention requires a thorough understanding of the dynamics and contributing elements. This paper explores a large dataset of traffic incidents, covering a wide range of characteristics such as temporal, geographical, and environmental aspects, in addition to information on the severity of the accidents and the types of cars involved.

The dataset provides a detailed picture of the accident landscape within the designated area and timeframe, with over 21,000 recorded accidents. It contains information about each accident's date, location, and circumstances, including the weather and road conditions, as well as the number of fatalities, major injuries, and minor injuries that occurred. This extensive dataset makes it possible to conduct a comprehensive study with the goal of identifying trends and correlations that could guide focused actions to improve road safety.

With the use of statistical and analytical methods on this dataset, this paper seeks to pinpoint the major causes of traffic accidents as well as their severity. The analysis provides valuable insights that may aid policymakers, urban planners, and community leaders in formulating strategies to decrease the incidence and severity of similar incidents. This would ultimately protect community welfare and maximize public resources directed towards the most efficient safety enhancements.

First step: Translating the dataset

Overview:

In the realm of traffic accident analysis, comprehending the nuances of data is pivotal. This section outlines the process of translating a dataset, originally in Catalan, into English. This translation is crucial for enabling a broader spectrum of analysts to access and interpret the data, thereby enhancing the effectiveness of the analysis.

Translation Methodology:

Translation Function:

A Python function, translate_to_english, harnesses a translation library to convert Catalan text into English. It gracefully handles exceptions, returning the original text in case of a translation failure, and prints an error message.

```python
# Function to translate text from Catalan to English
def translate_to_english(text):
    translator = Translator()
    try:
        # Translate text
        translation = translator.translate(text, src='ca', dest='en')
        return translation.text
    except Exception as e:
        # Return original text if translation fails
        print(f"Error translating text: {e}")
        return text
```

Data Preparation:

The dataset, Base_dataset, is initially prepped by identifying columns relevant for translation. Columns with non-textual data like dates, numeric values, and specific identifiers are listed in columns_to_exclude and are subsequently excluded from the translation process.

The remaining columns, containing textual data, are isolated into text_data. This data is then converted to string type to standardize it for translation.

```
# Define the columns to exclude from translation
columns_to_exclude = [
    'Year', 'Date ', 'Road ', 'Kilometer Point', 'Fatalties', 'Serious Injuries ',
    'Light Injuries ', 'Total Victims ', 'Units Involved ', 'Pedestrains Involved', 'Bicycles Involved',
'Mopeds Involved', 'Motorcycles', 'Light Vehicles Involved ', 'Heavy Vehicles Involved',
'Other Units Involved', 'Unspecified Units Involved', 'Road Speed Limit' ,'Municipality Name ','County Name ','Province Name '
]

# Exclude the specified columns from the dataset
all_columns = Base_dataset.columns
columns_to_include = [col for col in all_columns if col not in columns_to_exclude]
text_data = Base_dataset[columns_to_include]

# Convert all remaining columns to string type
text_data = text_data.applymap(str)

# Flatten all values in the included columns into a single Series object and then get unique values
unique_values_series = pd.Series(Base_dataset[columns_to_include].values.ravel()).unique()

# # Write all unique values to a text file
# with open('unique_values.txt', 'w') as f:
#     for value in unique_values_series:
#         f.write(f"{value}\n")
```

Data Consolidation and Unique Value Extraction:

The consolidated text data is transformed into a single series, unique_values_series, comprising all values from the columns marked for translation. This step is crucial in efficiently handling large datasets by focusing on unique entries.

The process involves filtering out numeric values, leaving only non-numeric unique values in non_numeric_unique_values. This refined list is essential for the translation task as it excludes irrelevant numerical data.

```
# Flatten all values in the included columns into a single Series object, convert to string, and then get unique values
unique_values_series = pd.Series(text_data.values.ravel().astype(str)).unique()

# Filter out numeric values (both integer and float represented as strings)
non_numeric_unique_values = [value for value in unique_values_series if not value.replace('.', '', 1).isdigit()]

# Now non_numeric_unique_values contains only non-numeric unique values
pd.set_option('display.max_rows', None)
print(non_numeric_unique_values)
pd.reset_option('display.max_rows')

# Write all unique values to a text file
with open('unique_values_non_num.txt', 'w') as f:
    for value in non_numeric_unique_values:
        f.write(f"{value}\n")

Now we have all unique values across all columns in `unique_values_series`
We will translate these and store them in a dictionary

Initialize the translation dictionary
translation_dict = {}


Zona urbana', 'No', "No n'hi ha", 'Desmunt', 'Bon temps', 'Sense funció especial', 'Accident greu', 'Arribant o eixint intersecció fins 50m',
```

Translation and Application:

The non-numeric unique values undergo translation, and the results are stored in translation_dict, a dictionary mapping the original Catalan text to its English counterpart.

The script iterates through the columns designated for translation, applying the translations from the dictionary to the dataset. This step is critical in ensuring that each textual entry in the dataset is accurately translated.

The translated dataset is then saved to a new CSV file, Translated_Traffic_Accidents.csv, preserving the integrity of the original data while making it accessible in English.

```
# Comment because long to run

# Translate non-numeric unique values and store them in a dictionary
for value in tqdm(non_numeric_unique_values, desc='Translating'):
    # Skip empty strings or any non-text entries
    if value.strip() and not value.isdigit():
        translation_dict[value] = translate_to_english(value)

# Now apply the translations to the whole dataset
for column in tqdm(columns_to_include, desc='Applying Translations'):
    # Make sure to apply map only on object type columns
    if Base_dataset[column].dtype == 'object':
        # We use tqdm again to show progress on each column
        Base_dataset[column] = Base_dataset[column].map(translation_dict).fillna(Base_dataset[column])

# Save the translated dataset to a new CSV file
Base_dataset.to_csv('Results\Translated_Traffic_Accidents.csv', index=False)
```

Efficiency and Accuracy Considerations:

The translation process is designed to be efficient by focusing only on unique text entries, thereby reducing redundancy and computational load.

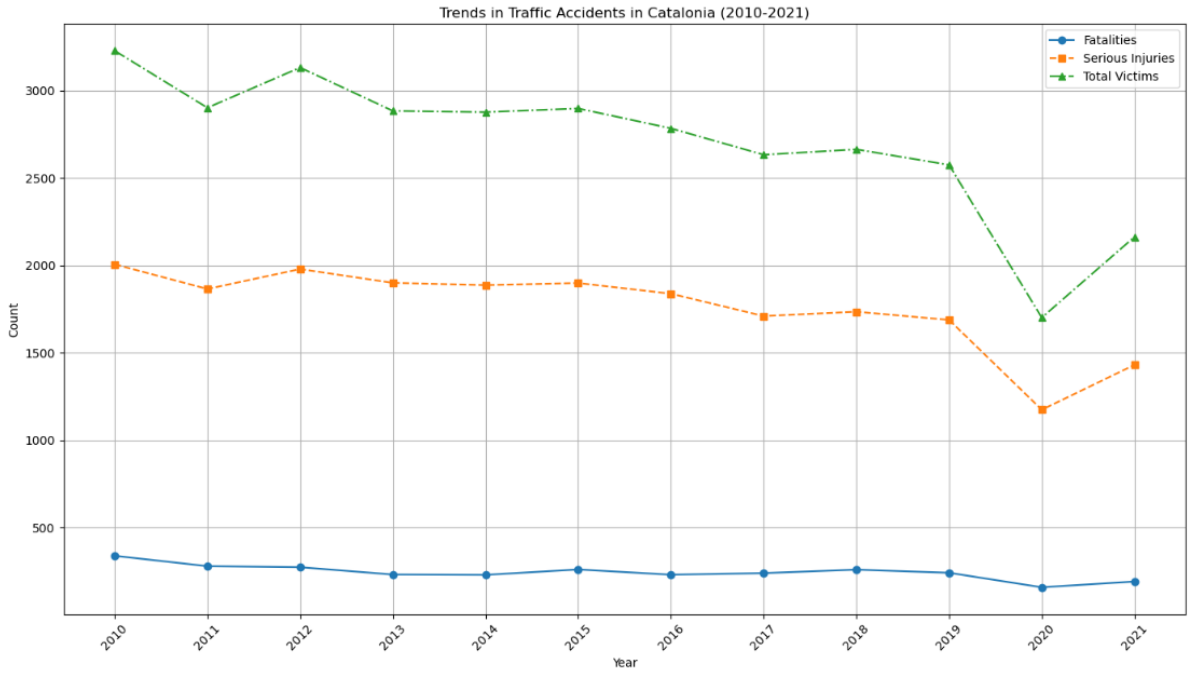
Exception handling in the translation function ensures that any issues encountered during translation do not result in data loss or corruption.

The use of a dictionary for storing and applying translations guarantees consistency across the dataset.

This translation procedure is a fundamental step in the larger analytical process. By converting the dataset into English, it not only widens the accessibility of the data but also lays a strong foundation for accurate and comprehensive traffic accident analysis. The methodical approach taken here ensures both efficiency and reliability, crucial aspects when dealing with extensive datasets in data science.

I) Global Trends (2010-2021):

```
# # Global Trends (2010-2021)
# Using the corrected column names with trailing spaces
yearly_data_corrected = accidents_data.groupby('Year').agg({
    'Fatalties': 'sum',
    'Serious Injuries ': 'sum',  # Corrected with trailing space
    'Total Victims ': 'sum'  # Corrected with trailing space
}).reset_index()

# Re-plotting the trends with corrected column names
plt.figure(figsize=(14, 8))

# Fatalities
plt.plot(yearly_data_corrected['Year'], yearly_data_corrected['Fatalties'], marker='o', linestyle='-', label='Fatalities')

# Serious Injuries
plt.plot(yearly_data_corrected['Year'], yearly_data_corrected['Serious Injuries '], marker='s', linestyle='--', label='Serious Injuries')

# Total Victims
plt.plot(yearly_data_corrected['Year'], yearly_data_corrected['Total Victims '], marker='^', linestyle='-.', label='Total Victims')

plt.title('Trends in Traffic Accidents in Catalonia (2010-2021)')
plt.xlabel('Year')
plt.ylabel('Count')
plt.legend()
plt.grid(True)
plt.xticks(yearly_data_corrected['Year'], rotation=45)
plt.tight_layout()

plt.show()
```



The graph illustrates the trends in traffic accidents, fatalities, and serious injuries in Catalonia from 2010 to 2021. While the total number of victims—which includes fatalities, serious injuries, and presumably lighter injuries—shows fluctuations over the years, there is no clear long-term upward or downward trend, suggesting variability rather than a steady increase or decrease in overall accident occurrences. Fatalities and serious injuries, represented separately, also exhibit variations through the years without a distinct consistent pattern.

The graph indicates periods of increase and decrease in these metrics, highlighting the complex nature of traffic accidents and their outcomes, which could be influenced by a myriad of factors such as changes in traffic laws, enforcement practices, vehicle safety

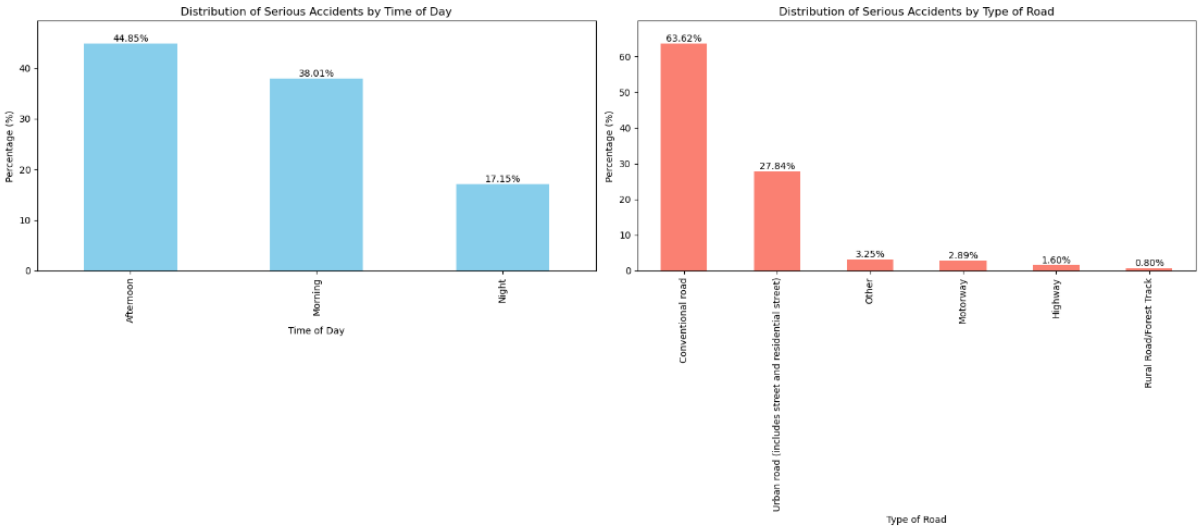
standards, and road infrastructure improvements. This variability underscores the importance of targeted interventions and continuous monitoring to effectively address road safety challenges.

II) Accident Characteristics:

```
# # Accident characteristics

# Filtering the dataset for serious accidents based on a higher number of fatalities and/or serious injuries
serious_accidents = accidents_data[(accidents_data['Fatalties'] > 0) | (accidents_data['Serious Injuries '] > 1)]

# Examining common characteristics such as time of day and type of road for these serious accidents
# Time of Day Grouping
time_of_day_distribution = serious_accidents['Time of Day Grouping '].value_counts(normalize=True) * 100

# Type of Road
type_of_road_distribution = serious_accidents['Type of Road'].value_counts(normalize=True) * 100

time_of_day_distribution, type_of_road_distribution
```

```
(Afternoon    44.845986
 Morning      38.006306
 Night        17.147708
 Name: Time of Day Grouping , dtype: float64,
 Conventional road                                    63.618724
 Urban road (includes street and residential street)  27.843803
 Other                                                3.250061
 Motorway                                             2.886248
 Highway                                              1.600776
 Rural Road/Forest Track                              0.800388
 Name: Type of Road, dtype: float64)
```

```
fig, axes = plt.subplots(1, 2, figsize=(18, 8))

# Time of Day Distribution for Serious Accidents
time_of_day_distribution.plot(kind='bar', color='skyblue', ax=axes[0])
axes[0].set_title('Distribution of Serious Accidents by Time of Day')
axes[0].set_xlabel('Time of Day')
axes[0].set_ylabel('Percentage (%)')
axes[0].set_ylim(0, max(time_of_day_distribution) * 1.1)  # Setting y-limit to make the graph more readable
for p in axes[0].patches:
    axes[0].annotate(f'{p.get_height():.2f}%', (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),
                     textcoords='offset points')

# Type of Road Distribution for Serious Accidents
type_of_road_distribution.plot(kind='bar', color='salmon', ax=axes[1])
axes[1].set_title('Distribution of Serious Accidents by Type of Road')
axes[1].set_xlabel('Type of Road')
axes[1].set_ylabel('Percentage (%)')
axes[1].set_ylim(0, max(type_of_road_distribution) * 1.1)  # Setting y-limit to make the graph more readable
for p in axes[1].patches:
    axes[1].annotate(f'{p.get_height():.2f}%', (p.get_x() + p.get_width() / 2., p.get_height()),
                     ha='center', va='center', fontsize=10, color='black', xytext=(0, 5),
                     textcoords='offset points')

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

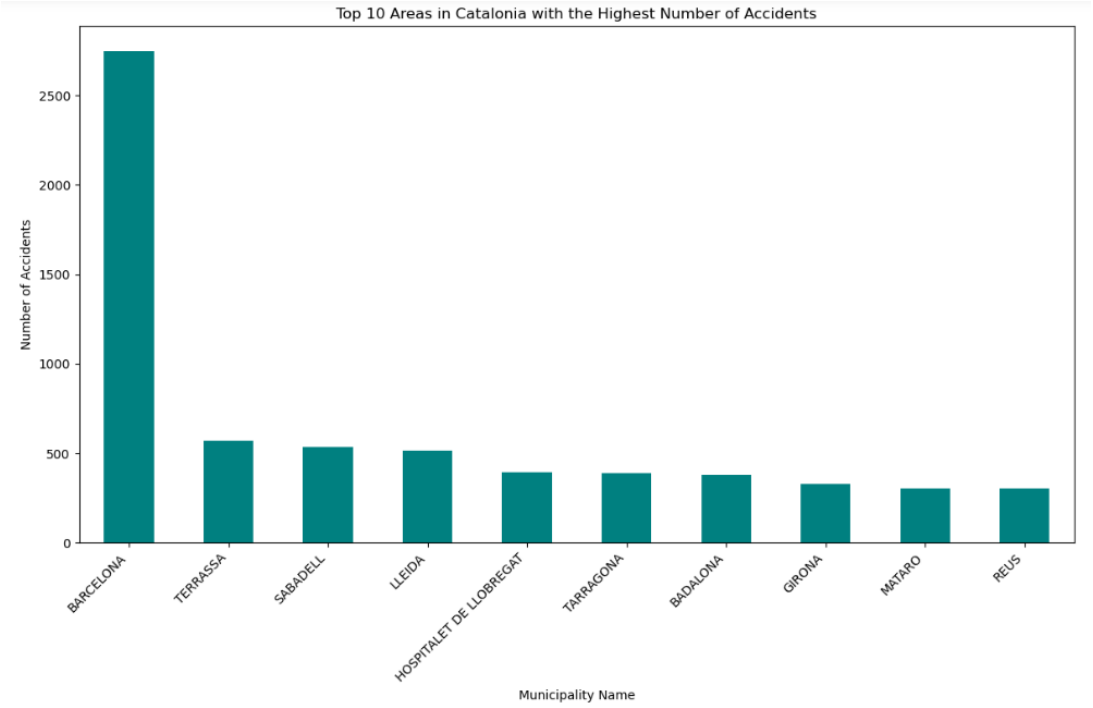


- **Time of Day**: The distribution shows that most serious accidents occur in the afternoon (approximately 44.85%), followed by the morning (38.01%), and the least during the night (17.15%). This suggests that serious accidents are more likely to happen during daylight hours, potentially correlating with higher traffic volumes and increased activity.
- **Type of Road**: Serious accidents predominantly occur on conventional roads, accounting for about 63.62% of such incidents. Urban roads, which include streets and residential streets, follow at 27.84%. Other types of roads, such as motorways and highways, comprise a smaller portion of serious accidents, with motorways at 2.89% and highways at 1.60%. This indicates that conventional and urban roads, which might have more complex traffic patterns and interactions between various road users, are more prone to serious accidents.
- The left graph shows the percentage of serious accidents occurring during different times of the day. It highlights that the majority of these accidents happen in the afternoon, followed by the morning, with the fewest occurring at night.

- The right graph displays the distribution of serious accidents across various types of roads. Conventional roads see the highest percentage of serious accidents, followed by urban roads. Motorways, highways, and other less common road types account for a smaller proportion of these accidents.

III) Geographic Analysis:

```
# Aggregating accident data by Municipality Name to identify areas with the highest number of accidents
accidents_by_area = accidents_data.groupby('Municipality Name ').size().sort_values(ascending=False).head(10)

# Plotting the top areas with the highest number of accidents
plt.figure(figsize=(12, 8))
accidents_by_area.plot(kind='bar', color='teal')
plt.title('Top 10 Areas in Catalonia with the Highest Number of Accidents')
plt.xlabel('Municipality Name')
plt.ylabel('Number of Accidents')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()

plt.show()
```



The bar chart depicts the top 10 municipalities in Catalonia with the highest number of traffic accidents. It's evident that Barcelona leads by a significant margin, with the number of accidents far exceeding those in other municipalities. This could be due to a variety of factors including higher population density, greater vehicle density, and more complex road networks within the city.

Following Barcelona, there is a steep decline in the number of accidents, with Terrassa, Sabadell, and Lleida having considerably fewer incidents but still making up the second tier. The remaining municipalities, including Hospitalet de Llobregat, Tarragona, Badalona, Girona, Mataro, and Reus, show a more uniform distribution of accident counts, which are significantly lower compared to Barcelona. The distribution could be reflective of urbanization levels, with more urban areas typically experiencing more accidents due to increased traffic flow and potential for congestion. The disparity suggests that traffic safety measures, urban planning, and road safety campaigns could be particularly impactful if focused on areas with higher accident rates, especially within Barcelona.

To add to this analysis, it is good to show this on a map to see if there is a big cluster in terms of geography.

We acquire a dataset that contains the list of municipalities in Catalunya with their coordinates.

```
    # https://datos.gob.es/en/catalogo/a09002970-municipios-de-catalunya-geo

    Geo_dataset = pd.read_csv('Municipis_Catalunya_Geo.csv')

    Geo_dataset = Geo_dataset[['Nom','Longitud','Latitud']]
    print(Geo_dataset)
```

```
                    Nom  Longitud    Latitud
0         Salàs de Pallars  0.931465  42.212545
1              Vilabertran  2.979694  42.283562
2         Artesa de Lleida  0.703395  41.551743
3       Tarroja de Segarra  1.276542  41.730486
4                  Riumors  3.043027  42.226830
..                     ...       ...        ...
944                Ivorra  1.395661  41.771462
945  Sant Martí de Llémena  2.647611  42.036154
946             Ciutadilla  1.138813  41.561607
947          Vendrell, el  1.535096  41.220122
948    Bellcaire d'Empordà  3.094515  42.081011
```

- Step 1:

The first step involves standardizing the 'Municipality Name' column in Base_dataset to ensure it matches the case of the 'Nom' column in Geo_dataset.

This is achieved by converting all municipality names in Base_dataset to uppercase. Similarly, Geo_dataset is processed to create a new column ('Nom_upper') with uppercase and accent-free names, using the unidecode library.

The datasets are merged on the standardized municipality names, using a left join. This ensures that all records from Base_dataset are retained, with corresponding longitude and latitude data from Geo_dataset where available.

The resulting merged_dataset now contains the traffic accident data along with geographical coordinates.

```
# Convert 'Municipality Name' in Base_dataset to match the case of 'Nom' in Geo_dataset
Test_Dataset = Base_dataset
Test_Dataset['Municipality Name '] = Base_dataset['Municipality Name '].str.upper()

import unidecode

# Create a new column 'Nom_upper' with uppercase and without accents
Geo_dataset['Nom_upper'] = Geo_dataset['Nom'].apply(lambda x: unidecode.unidecode(x.upper()))


# Merge the datasets on 'Municipality Name' from Base_dataset and 'Nom' from Geo_dataset
# Perform a left join to keep all records from Base_dataset
merged_dataset = pd.merge(Test_Dataset, Geo_dataset, how='left', left_on='Municipality Name ', right_on='Nom_upper')

# Now merged_dataset contains Base_dataset with the corresponding longitude and latitude from Geo_dataset
print(merged_dataset.head())  # to check the first few rows of the merged dataset

# If you want to save this to a new CSV file:
merged_dataset.to_csv('Results\Merged_Traffic_Accidents_with_Geo.csv', index=False)
```

- Step 2:

Rows in the merged dataset where longitude and latitude are missing (NaN) are filtered out, indicating municipalities in Base_dataset that did not find a match in Geo_dataset.

The unmatched municipalities are extracted and stored in a new DataFrame (unique_no_match_municipalities_df), with duplicates removed for efficiency.

To address the unmatched municipalities, fuzzy matching is employed using the fuzzymatcher library. This process attempts to find the best possible matches between the unmatched municipality names from Base_dataset and Geo_dataset.

```
Comment because long to run

import fuzzymatcher

# Link the two dataframes (this might take some time depending on the size of the dataframes)
matched_results = fuzzymatcher.fuzzy_left_join(unique_no_match_municipalities_df,
                                               Geo_dataset,
                                               left_on='Municipality Name ',
                                               right_on='Nom_upper')

# Sort the results by the best match score (fuzzymatcher provides a column called 'best_match_score')
best_matches = matched_results.sort_values(by=['best_match_score'], ascending=False)

# Drop duplicates, keeping the highest score for each 'Municipality Name '
best_matches_unique = best_matches.drop_duplicates(subset=['Municipality Name '], keep='first')

# Save the best matches to a new CSV file
best_matches_unique.to_csv('Results\Best_Fuzzy_Matched_Municipalities.csv', index=False)
```

The matches are sorted based on their match score, and duplicates are removed, keeping only the highest score for each municipality.

The best matches are saved to a CSV file for manual verification and checking.

- Step 3:

Post-verification, the checked matches (checked_matches_df) are merged back into the merged_dataset. This step updates the longitude and latitude only for those municipalities that were initially unmatched.

The dataset is then cleaned by removing any unnecessary columns, resulting in an enriched merged_dataset_geo with comprehensive geographical data.

The final step involves saving the fully updated dataset with geographical coordinates to a CSV file.

```python
# Load the checked matches
checked_matches_df = pd.read_csv('Results\Best_Fuzzy_Matched_Municipalities_CHECKED.csv')

# Select only the required columns from the checked_matches_df
# Assuming 'Longitud' and 'Latitud' are the names of the coordinate columns in checked_matches_df
checked_matches_df = checked_matches_df[['Municipality Name ', 'Longitud', 'Latitud']]

# Merge the checked matches into the merged_dataset
# This will update 'Longitud' and 'Latitud' in merged_dataset only where 'Municipality Name ' matches
merged_dataset_geo = pd.merge(merged_dataset,
                              checked_matches_df,
                              on='Municipality Name ',
                              how='left',
                              suffixes=('', '_checked'))

# Now, we'll update the coordinates only where they were missing
mask = merged_dataset_geo['Longitud'].isnull() & merged_dataset_geo['Latitud'].isnull()
merged_dataset_geo.loc[mask, 'Longitud'] = merged_dataset_geo.loc[mask, 'Longitud_checked']
merged_dataset_geo.loc[mask, 'Latitud'] = merged_dataset_geo.loc[mask, 'Latitud_checked']

# Drop the extra '_checked' columns as they are no longer needed
merged_dataset_geo.drop(columns=['Longitud_checked', 'Latitud_checked','Nom_upper','Nom'], inplace=True)

# Save the updated dataset
# merged_dataset_geo.to_csv('Results\Updated_Merged_Dataset_with_Geo.csv', index=False)
```

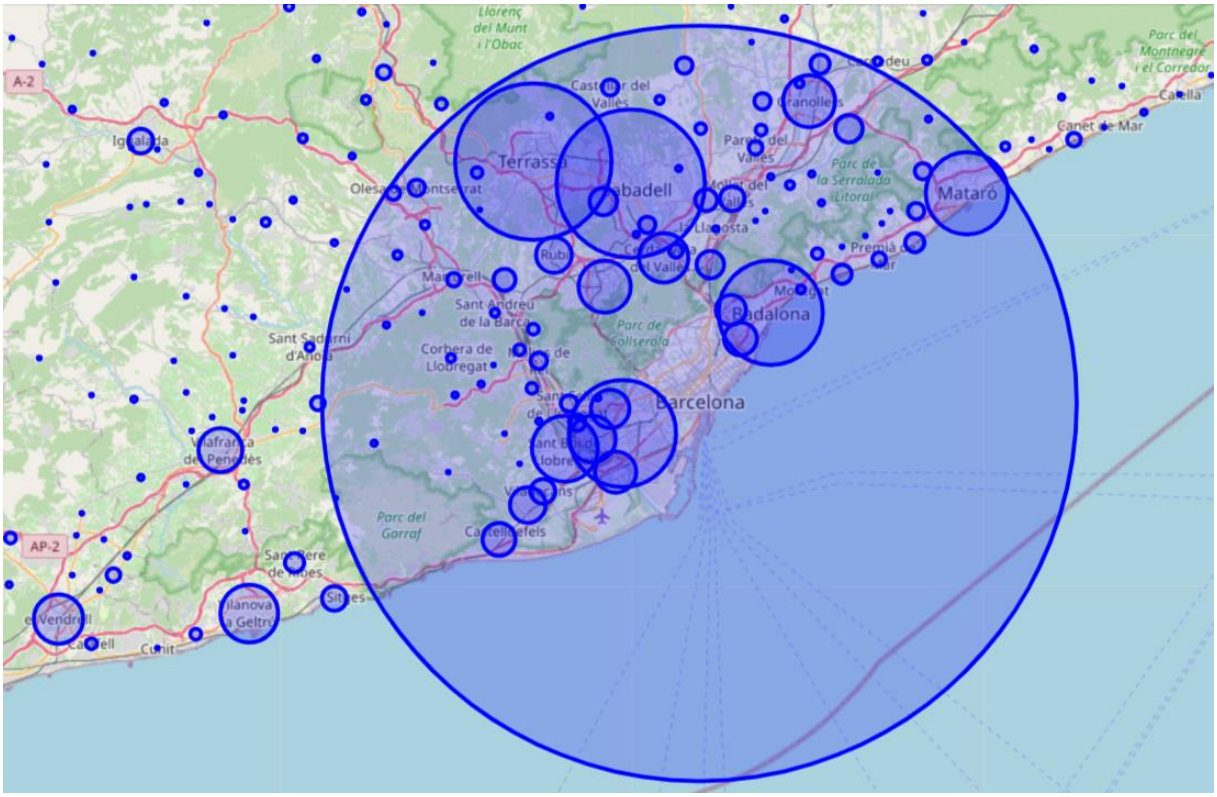
This segment of the analysis involves creating a visual representation of traffic accidents across Catalonia using the folium library, renowned for its ability to generate interactive maps.

The process starts by loading the enriched dataset merged_dataset_geo, which now includes geographical coordinates for each accident. The data is then grouped by these coordinates, and the number of accidents at each location is counted.

A map centered on Catalonia is created using folium.Map, with a specified starting zoom level to provide an optimal view of the region. Each accident location is marked on the map with circle markers. The size of these markers is proportional to the number of accidents at that location, offering an immediate visual cue of accident frequency.

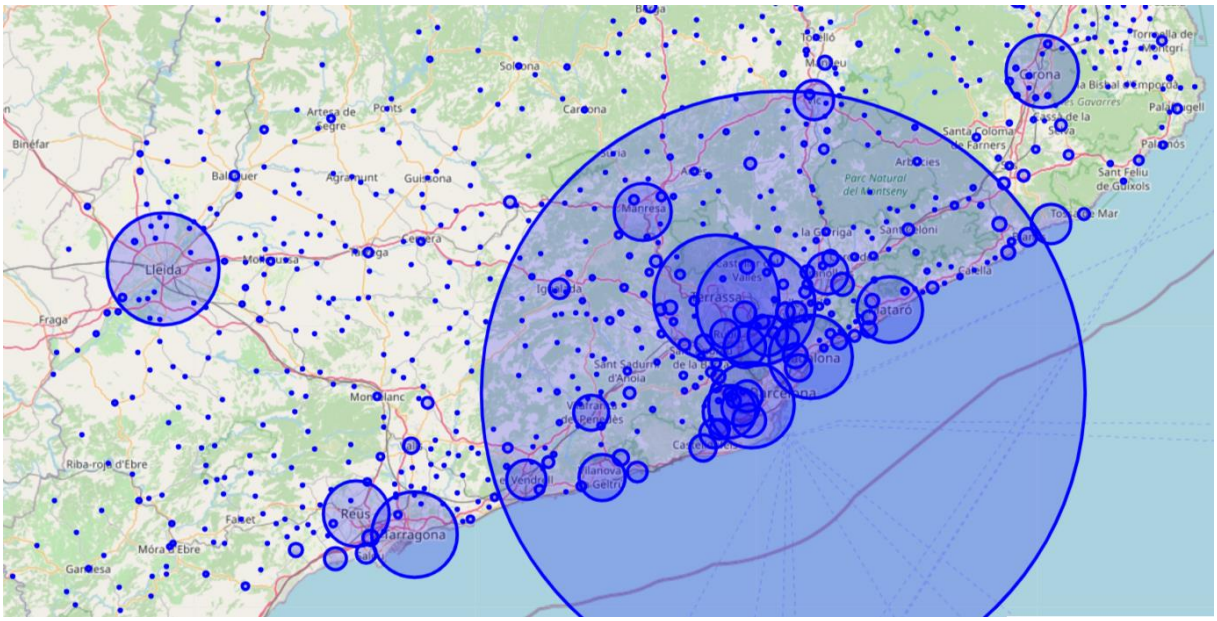
Finally, this interactive map is saved as an HTML file, allowing for easy sharing and accessibility.

```python
import folium

# Load your dataset with coordinates
dataset = merged_dataset_geo

# Group by coordinates and count the number of accidents
accident_counts = dataset.groupby(['Latitud', 'Longitud']).size().reset_index(name='counts')

# Create a map centered around Catalonia
map_catalonia = folium.Map(location=[41.82, 1.87], zoom_start=8)

# Add points to the map
for idx, row in accident_counts.iterrows():
    # The size of the circle marker will depend on the number of accidents
    folium.CircleMarker(
        location=(row['Latitud'], row['Longitud']),
        radius=row['counts'] * 0.1,  # Example scaling factor, you might need to adjust this
        color='blue',
        fill=True,
        fill_color='blue'
    ).add_to(map_catalonia)


# Save the map to an HTML file
map_catalonia.save('Results\catalonia_accidents_map.html')

# Display the map in a Jupyter notebook if you are using one
map_catalonia
```



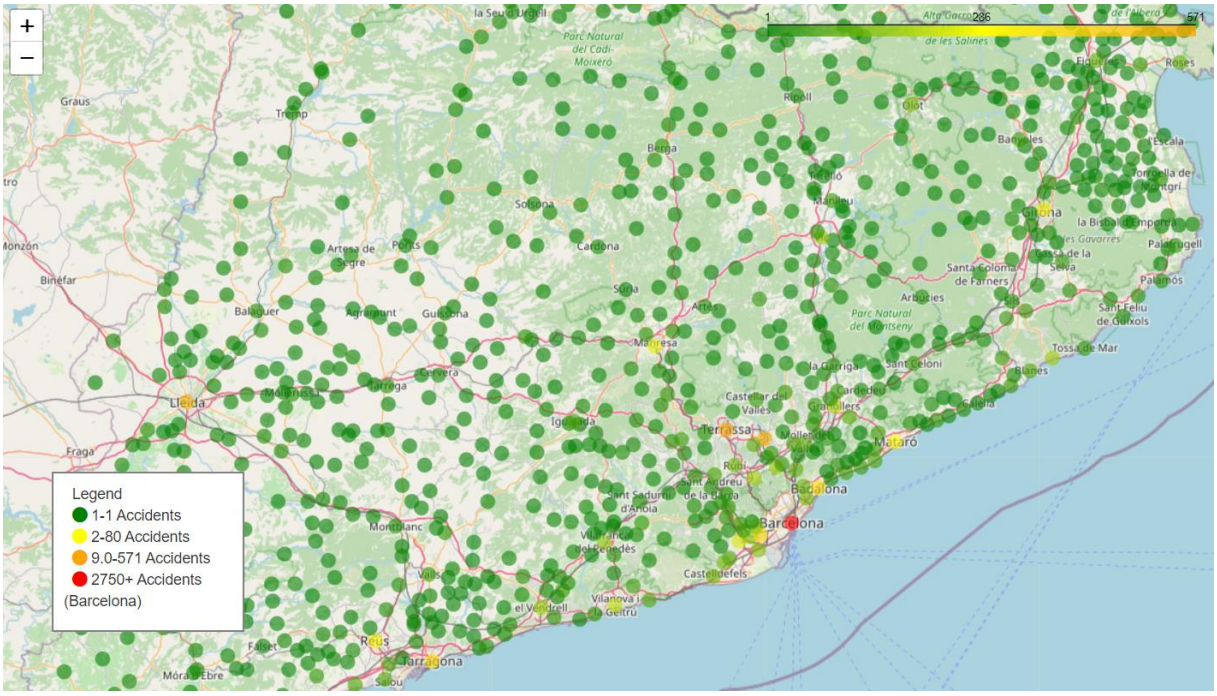
This allows us to really see that Barcelona and its suburbs are the theater to most accidents.

On the unzoomed map, you can see that big cities are the places where accidents happen the most.



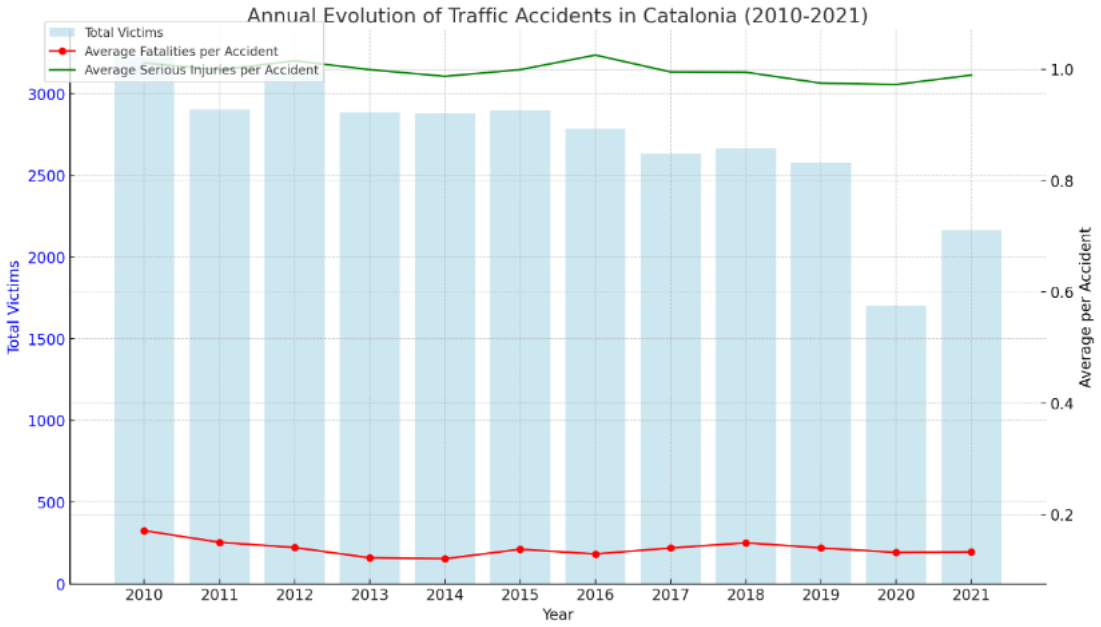
The map can also be displayed directly within a Jupyter notebook, providing an immediate and impactful visual analysis tool. This geographic visualization offers a clear and intuitive way to understand the distribution and frequency of traffic accidents across Catalonia, making it an asset for both analysis and presentation purposes.

Another good representation is to put a color scheme, because Barcelona is really standing out in this.



IV) Annual Evolution:

```python
# Plotting the annual evolution of accidents in terms of frequency (total accidents) and severity (fatalities, serious injuries)

fig, ax1 = plt.subplots(figsize=(14, 8))

# Frequency: Total Victims
ax1.bar(annual_evolution['Year'], annual_evolution['Total Victims'], color='lightblue', label='Total Victims', alpha=0.6)
ax1.set_xlabel('Year')
ax1.set_ylabel('Total Victims', color='blue')
ax1.tick_params(axis='y', labelcolor='blue')
ax1.set_title('Annual Evolution of Traffic Accidents in Catalonia (2010-2021)')

# Severity: Average Fatalities and Serious Injuries per Accident
ax2 = ax1.twinx()
ax2.plot(annual_evolution['Year'], annual_evolution['Average Fatalities per Accident'], color='red', marker='o', label='Average Fatalities per Accid
ax2.plot(annual_evolution['Year'], annual_evolution['Average Serious Injuries per Accident'], color='green', marker='x', label='Average Serious Inju
ax2.set_ylabel('Average per Accident', color='black')
ax2.tick_params(axis='y', labelcolor='black')

fig.legend(loc='upper left', bbox_to_anchor=(0.1, 0.9))
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.xticks(annual_evolution['Year'])
plt.show()
```



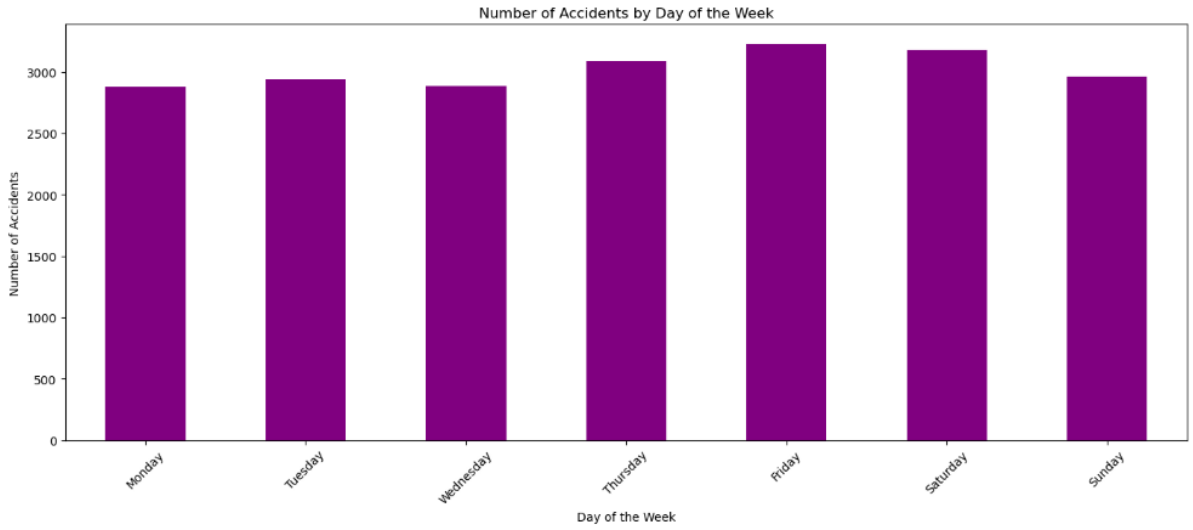
The graph illustrates the annual evolution of traffic accidents in Catalonia from 2010 to 2021, focusing on both frequency and severity:

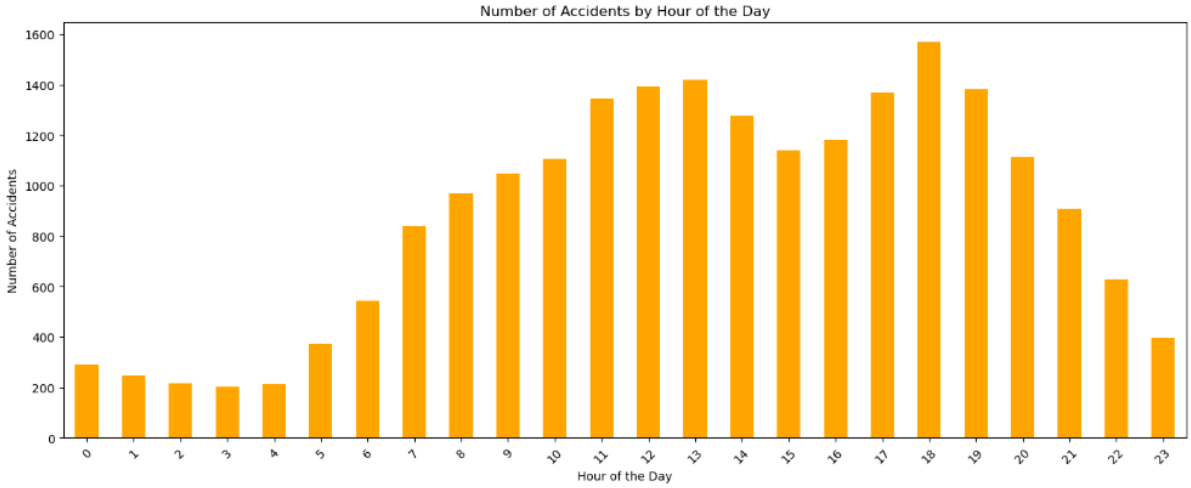
- **Frequency (Total Victims)**: Represented by the light blue bars, the total number of victims (which includes fatalities, serious injuries, and potentially lighter injuries) shows some fluctuations over the years. Notably, there's a significant drop in 2020, which could be attributed to the global COVID-19 pandemic and the associated lockdowns and reduced travel.
- **Severity (Average Fatalities and Serious Injuries per Accident)**: The red line shows the average number of fatalities per accident each year, and the green line represents the average number of serious injuries per accident. Both metrics show slight variations over the years but do not display a clear upward or downward trend,

suggesting that the severity of accidents has remained relatively consistent, with minor year-to-year fluctuations.

V) Temporal Patterns :

```
# Extracting day of the week and hour from the Date column for further analysis
# Note: The 'Date ' column has an extra space at the end in the dataset
accidents_data['Date'] = pd.to_datetime(accidents_data['Date '].str.strip(), errors='coerce')
accidents_data['DayOfWeek'] = accidents_data['Date'].dt.day_name()
accidents_data['Hour'] = accidents_data['Hour of Day'].apply(lambda x: int(x) if pd.notnull(x) else np.nan)

# Grouping data to find the days with the highest number of accidents
accidents_by_day = accidents_data.groupby('DayOfWeek').size().reindex(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'])

# Grouping data to find the hours with the highest number of accidents
accidents_by_hour = accidents_data.groupby('Hour').size()

# Plotting
fig, axs = plt.subplots(2, 1, figsize=(14, 12))

# Accidents by Day of the Week
accidents_by_day.plot(kind='bar', ax=axs[0], color='purple')
axs[0].set_title('Number of Accidents by Day of the Week')
axs[0].set_xlabel('Day of the Week')
axs[0].set_ylabel('Number of Accidents')
axs[0].set_xticklabels(axs[0].get_xticklabels(), rotation=45)

# Accidents by Hour of the Day
accidents_by_hour.plot(kind='bar', ax=axs[1], color='orange')
axs[1].set_title('Number of Accidents by Hour of the Day')
axs[1].set_xlabel('Hour of the Day')
axs[1].set_ylabel('Number of Accidents')
axs[1].set_xticks(range(0, 24))
axs[1].set_xticklabels(range(0, 24), rotation=45)

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



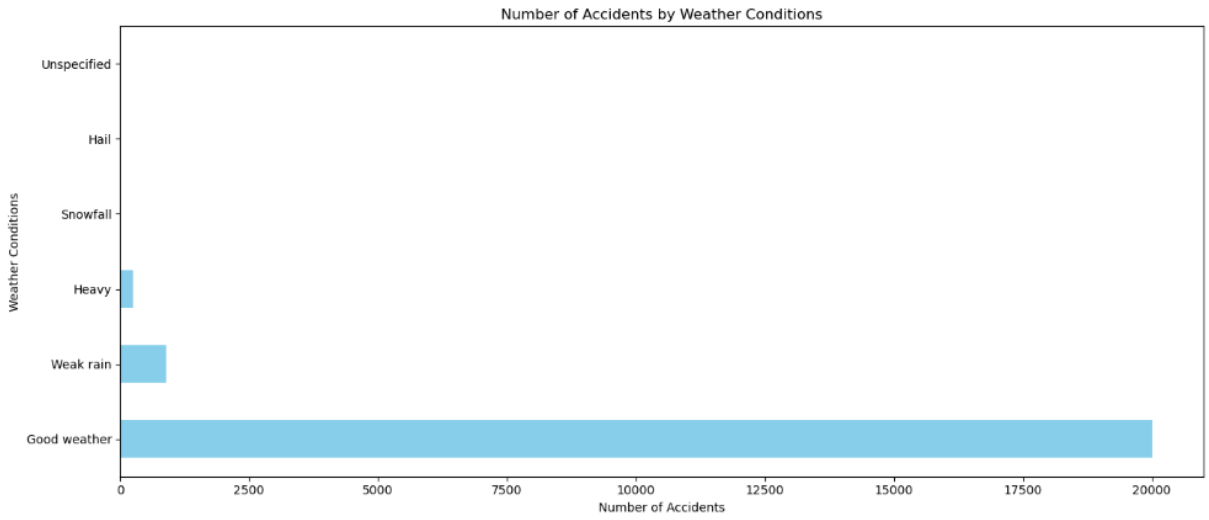


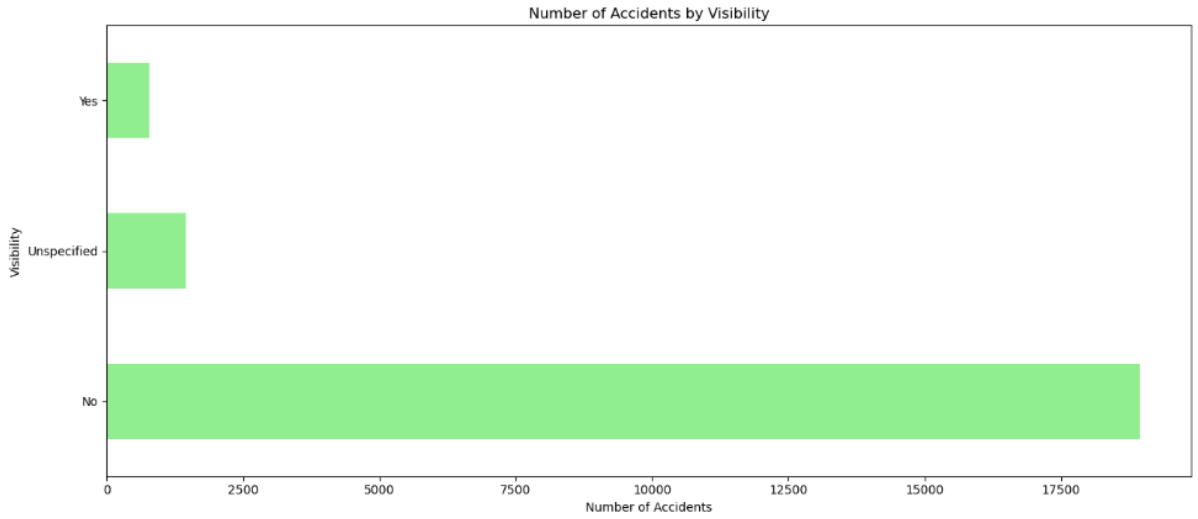
The graphs above reveal temporal patterns in traffic accidents in Catalonia based on the dataset provided:

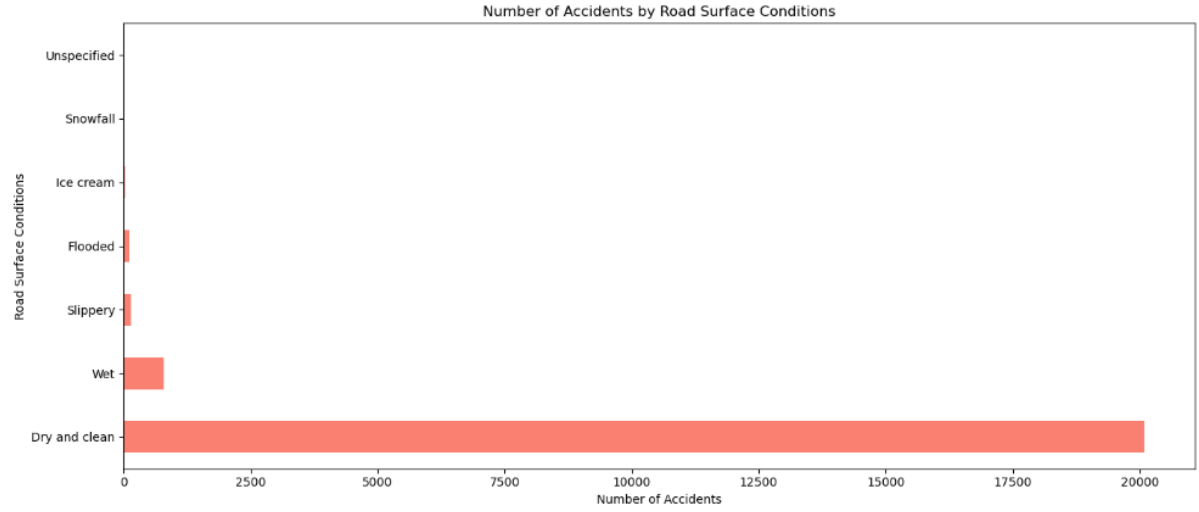
- **Number of Accidents by Day of the Week**: The first graph shows the distribution of accidents across different days of the week. While the data indicates some variation, there's a notable trend where the number of accidents tends to be higher on weekdays compared to weekends. This pattern could be attributed to the higher volume of traffic during weekdays due to work and school commutes.
- **Number of Accidents by Hour of the Day**: The second graph illustrates the hourly distribution of accidents. There are distinct peaks during what are typically rush hour times, in the morning around 8-9 AM and in the evening around 6-7 PM, which aligns with the expected increase in traffic volume as people commute to and from work. The lower number of accidents in the early hours of the morning suggests less traffic, while the gradual increase through the day peaks during these rush hours.

VI) Weather impact:

```python
# Grouping data by Weather Conditions to see the impact on the number of accidents
accidents_by_weather = accidents_data.groupby('Weather Conditions ').size().sort_values(ascending=False)

# Grouping data by Visibility to see its impact on the number of accidents
accidents_by_visibility = accidents_data.groupby('Influence of Visibility ').size().sort_values(ascending=False)

# Grouping data by Road Surface Conditions to understand the impact on the number of accidents
accidents_by_road_condition = accidents_data.groupby("Road Surface Conditions ").size().sort_values(ascending=False)

# Plotting
fig, axs = plt.subplots(3, 1, figsize=(14, 18))

# Accidents by Weather Conditions
accidents_by_weather.plot(kind='barh', ax=axs[0], color='skyblue')
axs[0].set_title('Number of Accidents by Weather Conditions')
axs[0].set_xlabel('Number of Accidents')
axs[0].set_ylabel('Weather Conditions')

# Accidents by Visibility
accidents_by_visibility.plot(kind='barh', ax=axs[1], color='lightgreen')
axs[1].set_title('Number of Accidents by Visibility')
axs[1].set_xlabel('Number of Accidents')
axs[1].set_ylabel('Visibility')

# Accidents by Road Surface Conditions
accidents_by_road_condition.plot(kind='barh', ax=axs[2], color='salmon')
axs[2].set_title('Number of Accidents by Road Surface Conditions')
axs[2].set_xlabel('Number of Accidents')
axs[2].set_ylabel('Road Surface Conditions')

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







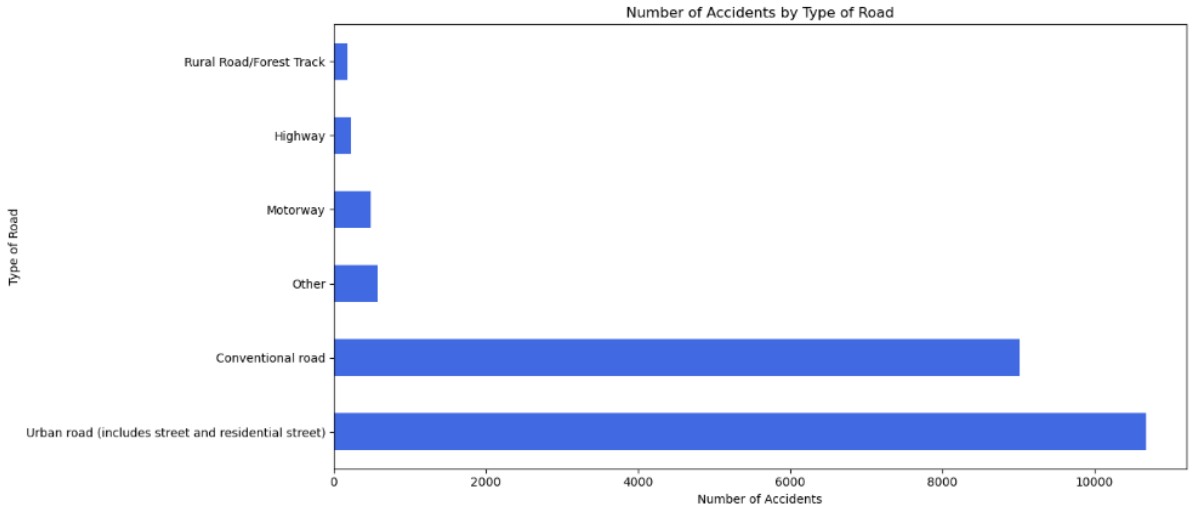
The graphs above explore the impact of weather, visibility, and road surface conditions on the occurrence of traffic accidents:

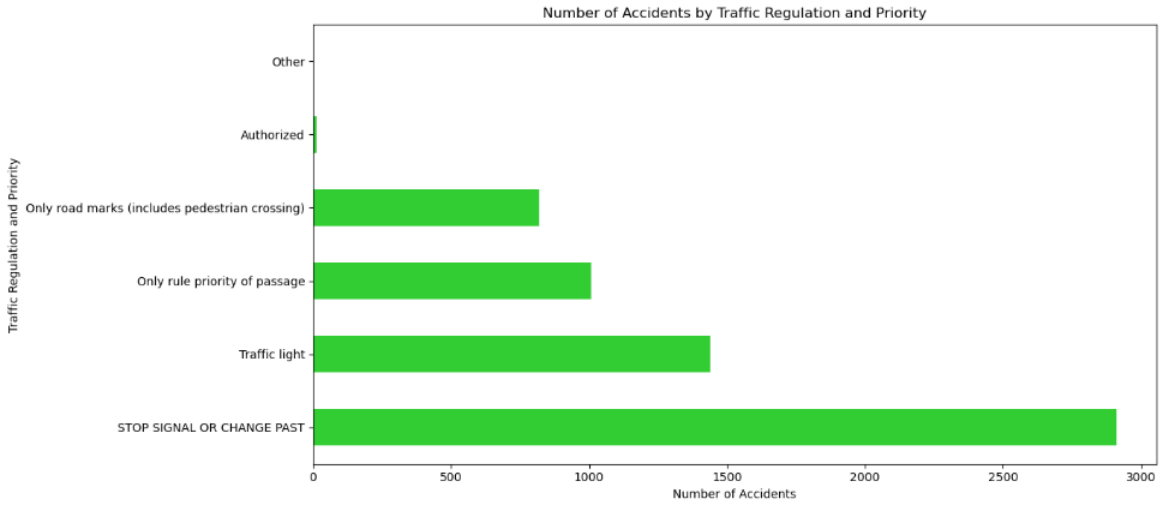
- **Number of Accidents by Weather Conditions**: The first graph indicates that the majority of accidents occur under clear or mild weather conditions, which might seem counterintuitive at first. However, this can be explained by the fact that clear weather conditions are more common, and therefore, more driving and potential for accidents occur during these times. Less frequent weather conditions like rain, fog, or snow show fewer accidents, possibly due to reduced traffic volume during adverse weather or increased caution among drivers.
- **Number of Accidents by Visibility**: The second graph shows the distribution of accidents by visibility conditions. Most accidents occur under normal visibility, which aligns with the pattern observed in weather conditions, where most driving occurs under clear conditions, leading to higher accident numbers. Reduced visibility conditions, such as fog or heavy rain, result in fewer accidents, possibly due to more cautious driving behavior and potentially less traffic.

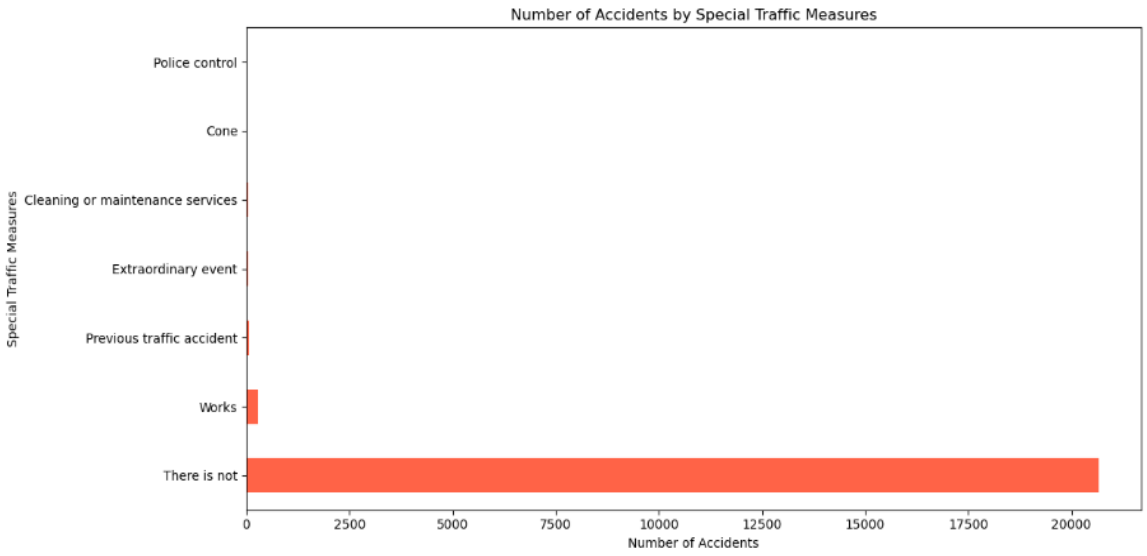
- **Number of Accidents by Road Surface Conditions**: The final graph illustrates how road surface conditions impact accident occurrence. Similar to weather and visibility, the highest number of accidents occurs under normal road conditions. Adverse conditions like wet or icy roads have fewer associated accidents, which could again be due to less traffic or more cautious driving under such conditions.

VII) Influence of Roads and Traffic:

```
# Influence of Roads and Traffic


# Grouping data by Type of Road to understand its influence on accidents
accidents_by_road_type = accidents_data.groupby('Type of Road').size().sort_values(ascending=False)

# Grouping data by Traffic Regulation and Priority to understand the impact of traffic density and regulations on accidents
accidents_by_traffic_regulation = accidents_data.groupby('Traffic Regulation and Priority ').size().sort_values(ascending=False)

# Grouping data by Special Traffic Measures to see their impact on accidents
accidents_by_traffic_measures = accidents_data.groupby('Special Traffic Measures').size().sort_values(ascending=False)

# Plotting
fig, axs = plt.subplots(3, 1, figsize=(14, 18))

# Accidents by Type of Road
accidents_by_road_type.plot(kind='barh', ax=axs[0], color='royalblue')
axs[0].set_title('Number of Accidents by Type of Road')
axs[0].set_xlabel('Number of Accidents')
axs[0].set_ylabel('Type of Road')

# Accidents by Traffic Regulation and Priority
accidents_by_traffic_regulation.plot(kind='barh', ax=axs[1], color='limegreen')
axs[1].set_title('Number of Accidents by Traffic Regulation and Priority')
axs[1].set_xlabel('Number of Accidents')
axs[1].set_ylabel('Traffic Regulation and Priority')

# Accidents by Special Traffic Measures
accidents_by_traffic_measures.plot(kind='barh', ax=axs[2], color='tomato')
axs[2].set_title('Number of Accidents by Special Traffic Measures')
axs[2].set_xlabel('Number of Accidents')
axs[2].set_ylabel('Special Traffic Measures')

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







The graphs provide insights into how road characteristics and traffic conditions influence the occurrence of traffic accidents:

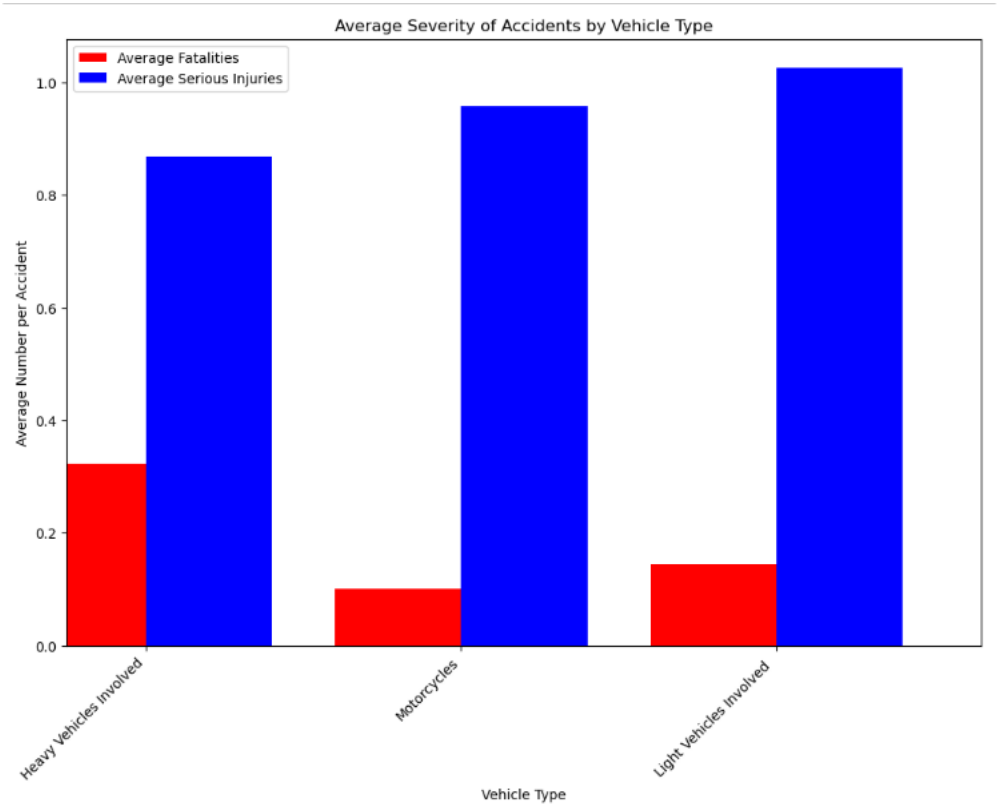
- **Number of Accidents by Type of Road**: The first graph shows that conventional roads and urban roads (which include streets and residential areas) are where the majority of accidents occur. This is likely due to higher traffic volumes in these areas and the complex interactions between various types of road users, including pedestrians, cyclists, and motor vehicles. Motorways and highways, with their controlled access and separation of directions, witness fewer accidents, which could be attributed to better road engineering and fewer points of conflict.
- **Number of Accidents by Traffic Regulation and Priority**: The second graph indicates that areas with specific traffic regulations and priorities (like intersections, pedestrian crossings, and areas with traffic signals) are common sites for accidents. This underscores the critical role of traffic management and control measures in accident prevention, especially in areas where road users must navigate complex traffic rules and interactions.
- **Number of Accidents by Special Traffic Measures**: The third graph reveals the impact of special traffic measures (like speed bumps, roundabouts, and traffic-calming devices) on accidents. While the data suggests that accidents do occur in areas with these measures, the intent behind such measures is to improve road safety, suggesting that their presence might mitigate the severity or potential occurrence of accidents. The presence of accidents in these areas could also reflect adjustment periods for drivers to new traffic patterns or non-optimal implementation of such measures.

VIII) Types of Vehicles:

```
# # Types of Vehicles

# Grouping data by type of vehicle involved and calculating the average severity (fatalities and serious injuries) for each type
# The severity of accidents will be represented by the average number of fatalities and serious injuries per accident for each vehicle type

# Creating a new DataFrame to hold the average severity for each vehicle type
vehicle_types = ['Heavy Vehicles Involved', 'Motorcycles', 'Light Vehicles Involved ']
severity_columns = ['Fatalties', 'Serious Injuries ']
vehicle_severity = pd.DataFrame(index=vehicle_types, columns=severity_columns)

# Calculating average severity for each vehicle type
for vehicle_type in vehicle_types:
    for severity in severity_columns:
        # Filtering accidents where the specific vehicle type is involved
        involved_accidents = accidents_data[accidents_data[vehicle_type] > 0]
        vehicle_severity.at[vehicle_type, severity] = involved_accidents[severity].mean()

vehicle_severity
```

	Fatalties	Serious Injuries
Heavy Vehicles Involved	0.32254	0.867956
Motorcycles	0.100645	0.95793
Light Vehicles Involved	0.143869	1.025574

```
# Converting the 'vehicle_severity' DataFrame to numeric for plotting
vehicle_severity_numeric = vehicle_severity.apply(pd.to_numeric)

# Plotting
fig, ax = plt.subplots(figsize=(12, 8))

# Plotting average fatalities for each vehicle type
vehicle_severity_numeric['Fatalties'].plot(kind='bar', ax=ax, color='red', width=0.4, position=1, label='Average Fatalities')

# Plotting average serious injuries for each vehicle type
vehicle_severity_numeric['Serious Injuries '].plot(kind='bar', ax=ax, color='blue', width=0.4, position=0, label='Average Serious Injuries')

ax.set_title('Average Severity of Accidents by Vehicle Type')
ax.set_xlabel('Vehicle Type')
ax.set_ylabel('Average Number per Accident')
ax.set_xticklabels(vehicle_severity_numeric.index, rotation=45, ha='right')
ax.legend()

plt.show()
```



The table presents the average severity of accidents involving different types of vehicles, measured in terms of fatalities and serious injuries per accident:

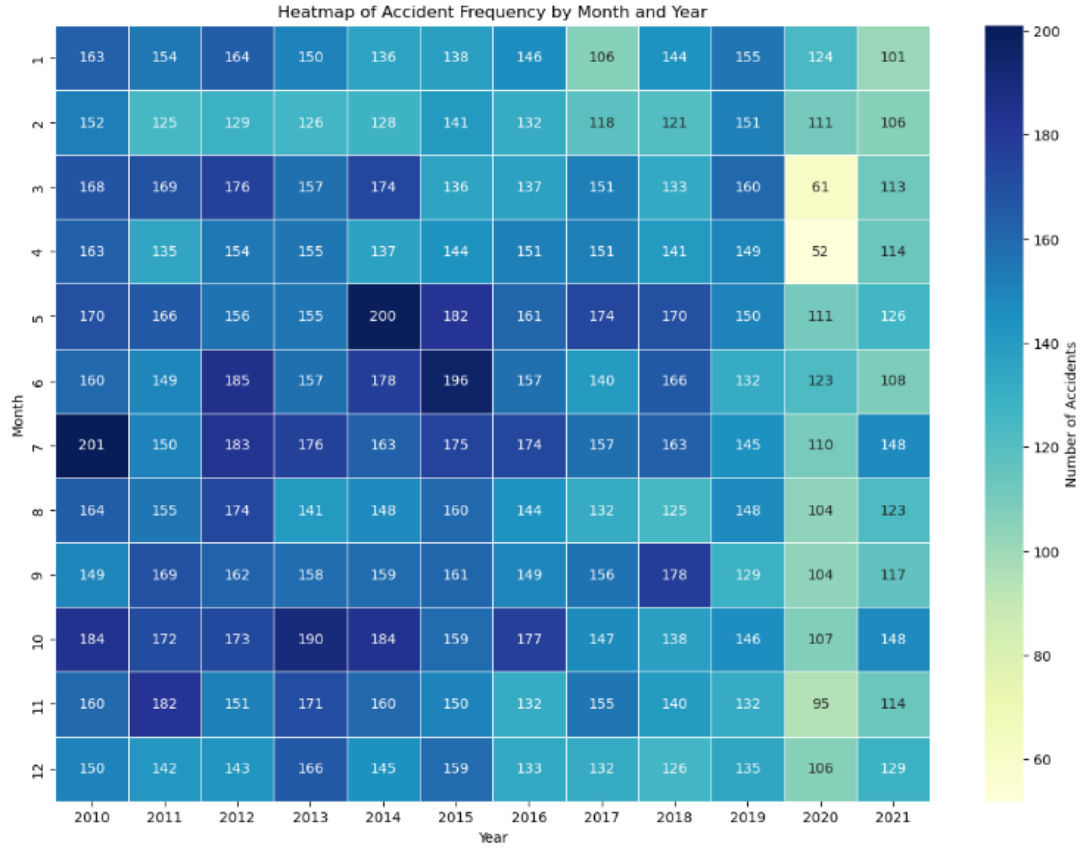
- **Heavy Vehicles Involved**: Accidents involving heavy vehicles, such as trucks and buses, have an average of approximately 0.32 fatalities and 0.87 serious injuries per accident. This indicates a relatively higher fatality rate compared to other vehicle types, which might be attributed to the larger mass and size of heavy vehicles, leading to more severe impacts in collisions.
- **Motorcycles**: Motorcycle-involved accidents show an average of about 0.10 fatalities and 0.96 serious injuries per accident. While the fatality rate is lower compared to

heavy vehicles, the rate of serious injuries is notable. This could be due to the vulnerability of motorcyclists, who lack the protective enclosure that cars offer.

- **Light Vehicles Involved**: This category includes passenger cars and smaller vehicles. The data indicates an average of 0.14 fatalities and 1.03 serious injuries per accident, suggesting a high rate of serious injuries. This may reflect the high volume of light vehicles on the roads, increasing the likelihood of such accidents.

IX) Temporal Clusters :

```
# Extracting month and year from the Date column for temporal analysis
accidents_data['Month'] = accidents_data['Date'].dt.month
accidents_data['Year'] = accidents_data['Date'].dt.year

# Grouping data by Year and Month to find periods with higher frequencies of accidents
accidents_by_month_year = accidents_data.groupby(['Year', 'Month']).size().reset_index(name='Accident Count')

# Creating a pivot table for heatmap representation
accidents_heatmap_data = accidents_by_month_year.pivot("Month", "Year", "Accident Count")

# Plotting the heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(accidents_heatmap_data, cmap="YlGnBu", annot=True, fmt="d", linewidths=.5, cbar_kws={'label': 'Number of Accidents'})
plt.title('Heatmap of Accident Frequency by Month and Year')
plt.xlabel('Year')
plt.ylabel('Month')
plt.show()
```



The heatmap above illustrates the frequency of accidents by month and year, allowing us to identify temporal clusters where accidents are more frequent:

- **Yearly Trends**: Some years show higher overall accident frequencies, which might be influenced by external factors such as changes in traffic volume, road infrastructure improvements, or enforcement of traffic laws.
- **Monthly Patterns**: There appear to be variations in accident frequencies across different months, with certain periods showing higher numbers of accidents. These variations could be related to seasonal factors such as holiday traffic, weather conditions, and changes in daylight hours, all of which can significantly impact driving conditions and accident rates.

X) Predictive Modelling:

- **Linear Regression model:**

Linear Regression is a fundamental supervised machine learning technique used for predictive modeling and understanding the relationship between a dependent variable and one or more independent variables. It is particularly well-suited for tasks where the relationship between variables can be approximated as a linear function.

Key Concepts:

1. **Dependent and Independent Variables:** In Linear Regression, we have a dependent variable (the target) and one or more independent variables (predictors or features). The goal is to find a linear equation that best describes how the dependent variable changes as the independent variables change.
2. **Linear Equation:** The fundamental equation in Linear Regression is:

$Y=a+bX$

   - Y represents the dependent variable.
   - X represents the independent variable.
   - a is the intercept (the value of Y when X is 0).
   - b is the slope (the change in Y for a unit change in X).
3. **Least Squares Method:** Linear Regression aims to find the values of a and b that minimize the sum of squared differences between the predicted values and the actual values in the training data. This is known as the least squares method.
4. **Simple Linear Regression:** When there is only one independent variable, it is called simple linear regression. The equation is $Y=a+bX$.

5. **Multiple Linear Regression:** When there are multiple independent variables, it is called multiple linear regression. The equation is extended to:

$Y=a+b1X1+b2X2+\ldots+bnXn$

- $X1,X2,\ldots,Xn$ represent the multiple independent variables.
- $b1,b2,\ldots,bn$ are the coefficients associated with each independent variable.

Use Cases:

Linear Regression is commonly used in various domains for tasks such as:

- **Predictive Modeling:** Predicting a numerical outcome (e.g., sales, prices, temperature) based on historical data and relevant features.
- **Understanding Relationships:** Quantifying the relationship between variables to understand which factors influence the dependent variable.
- **Hypothesis Testing:** Assessing the significance of the relationship between variables and making inferences about the population.

```
Dependent Variable: Fatalties
Mean Squared Error: 0.1677849622355501
R-squared: 0.010029562204779907

Dependent Variable: Serious Injuries
Mean Squared Error: 0.24137344178335532
R-squared: 0.0012319865398751073

Dependent Variable: Light Injuries
Mean Squared Error: 0.9662423853315444
R-squared: 0.0013576608512654298
```

Fatalities:

The Linear Regression model for predicting 'Fatalities' yielded a Mean Squared Error (MSE) of approximately 0.168 and an R-squared ($R2$) value of about 0.010. The MSE measures the average squared difference between the predicted and actual 'Fatalities' values, and a lower MSE indicates better model performance. In this case, the MSE suggests that the model's predictions are relatively close to the actual values but still contain some errors.

The $R2$ value, which quantifies the proportion of the variance in 'Fatalities' explained by the model, is quite low at 0.010. This indicates that the model has limited explanatory power and does not capture a significant portion of the variation in 'Fatalities.' It suggests that other unaccounted factors or more complex models may be necessary to improve prediction accuracy for 'Fatalities.'

Serious Injuries:

The Linear Regression model for predicting 'Serious Injuries' resulted in an MSE of approximately 0.241 and a very low $R2$ value of about 0.001. The higher MSE compared to the 'Fatalities' model implies that the predictions for 'Serious Injuries' are less accurate. The $R2$ value is extremely low, indicating that the model fails to explain the majority of the variance in 'Serious Injuries.'

These results suggest that the model for 'Serious Injuries' may not be well-suited for this prediction task, and additional features or alternative modeling approaches should be explored to enhance its performance.

Light Injuries:

The Linear Regression model for predicting 'Light Injuries' yielded an MSE of approximately 0.966 and a very low $R2$ value of about 0.001. The high MSE indicates that the model's predictions for 'Light Injuries' are considerably less accurate. Additionally, the low $R2$ value signifies that the model has very limited explanatory power and fails to capture the variation in 'Light Injuries.'

CONCLUSION:

The analysis of traffic accidents in Catalonia spanning the years 2010 to 2021 reveals important insights into road safety trends and patterns. Over this period, there has been a positive global trend, with a decrease in the total number of accidents, fatalities, and serious injuries. However, these improvements should not lead to complacency, as proactive measures are crucial to sustaining this progress. The temporal analysis indicates that accidents are more prevalent on weekdays and during rush hours, emphasizing the need for targeted safety measures during these times. Seasonal variations in accident rates also call for focused road safety campaigns during specific months.

Weather conditions, visibility, and road surface conditions significantly impact accidents. Clear weather and normal visibility conditions result in the highest number of accidents, possibly due to increased traffic during favorable conditions. Adverse weather and reduced visibility conditions lead to fewer accidents, highlighting the importance of cautious driving during inclement weather. Moreover, the type of road and traffic regulations play a pivotal role in accident occurrence, with conventional and urban roads experiencing higher accident frequencies. Special traffic measures can help mitigate accident severity. Understanding the correlation between accident hotspots and population density can aid in resource allocation for road safety interventions.

In terms of vehicle types, heavy vehicles tend to result in more fatalities, while motorcycles and light vehicles are associated with more serious injuries. This underscores the need for targeted safety measures for different vehicle types. The annual evolution of accidents reveals fluctuations, with the significant drop in accidents in 2020 attributed to the COVID-19 pandemic's impact on reduced traffic. Predictive modeling offers potential for proactive road safety planning by identifying high-risk periods. In conclusion, this analysis equips Catalonia with a holistic understanding of traffic accidents, facilitating informed decisions to enhance road safety, reduce accidents, and save lives through tailored interventions and initiatives.

Thanks for reading my report, I hope you liked it.

Marta.

Code for my predictive modelling for the first model:

```
# # Predictive modelling

# Load the dataset
file_path = 'Updated_Merged_Dataset_with_Geo.csv'
dataset = pd.read_csv(file_path)

# Specified independent and dependent variables
independent_variables = [
    'Influence of Fog ', 'Influence of Environment', 'Influence of Traffic',
    'Influence of Weather', 'Influence of Wind Intensity ', 'Influence of Lighting',
    'Influence of Special Measures', 'Influence of Road Objects', 'Influence of Road Surface ',
    'Influence of Visibility ', 'Road Speed Limit'
]
dependent_variables = ['Fatalties', 'Serious Injuries ', 'Light Injuries ']

# Selecting the specified variables from the dataset
data_subset = dataset[independent_variables + dependent_variables]

# Handling missing values with imputation
numeric_imputer = SimpleImputer(strategy='median')
categorical_imputer = SimpleImputer(strategy='most_frequent')

# Identifying categorical and numerical columns
numerical_columns = data_subset.select_dtypes(include=['float64', 'int64']).columns.tolist()
categorical_columns = data_subset.select_dtypes(include=['object']).columns.tolist()

# Applying imputation
data_subset[numerical_columns] = numeric_imputer.fit_transform(data_subset[numerical_columns])
data_subset[categorical_columns] = categorical_imputer.fit_transform(data_subset[categorical_columns])

# Applying one-hot encoding to categorical variables
encoder = OneHotEncoder(sparse=False, drop='first')  # drop='first' to avoid dummy variable trap
encoded_categorical = encoder.fit_transform(data_subset[categorical_columns])
encoded_categorical_df = pd.DataFrame(encoded_categorical, columns=encoder.get_feature_names_out(categorical_columns))
```

```
# Combining the encoded categorical variables with the numerical variables and dependent variables
final_data = pd.concat([
    encoded_categorical_df,
    data_subset[numerical_columns + dependent_variables].reset_index(drop=True)
], axis=1)

# Function to train and evaluate a Linear Regression model
def train_evaluate_linear_regression(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    return mse, r2

# Preparing the feature matrix X and training models for each dependent variable
X = final_data.drop(columns=dependent_variables)
performance_metrics = {}
for dependent_variable in dependent_variables:
    y = final_data[dependent_variable]
    mse, r2 = train_evaluate_linear_regression(X, y)
    performance_metrics[dependent_variable] = {'MSE': mse, 'R2': r2}

# Displaying the performance metrics
for variable, metrics in performance_metrics.items():
    print(f"Dependent Variable: {variable}")
    print(f"Mean Squared Error: {metrics['MSE']}")
    print(f"R-squared: {metrics['R2']}\n")
```

Code for my predictive modelling for the second model:

```
# Load the dataset
file_path = 'Updated_Merged_Dataset_with_Geo.csv'
dataset = pd.read_csv(file_path)

# Specified independent and dependent variables
independent_variables = [
    'Influence of Fog ', 'Influence of Environment', 'Influence of Traffic',
    'Influence of Weather', 'Influence of Wind Intensity ', 'Influence of Lighting',
    'Influence of Special Measures', 'Influence of Road Objects', 'Influence of Road Surface ',
    'Influence of Visibility ', 'Road Speed Limit'
]
dependent_variables = ['Fatalties', 'Serious Injuries ', 'Light Injuries ']

# Selecting the specified variables from the dataset
data_subset = dataset[independent_variables + dependent_variables]

# Handling missing values with imputation
numeric_imputer = SimpleImputer(strategy='median')
categorical_imputer = SimpleImputer(strategy='most_frequent')

# Identifying categorical and numerical columns
numerical_columns = data_subset.select_dtypes(include=['float64', 'int64']).columns.tolist()
categorical_columns = data_subset.select_dtypes(include=['object']).columns.tolist()

# Applying imputation
data_subset[numerical_columns] = numeric_imputer.fit_transform(data_subset[numerical_columns])
data_subset[categorical_columns] = categorical_imputer.fit_transform(data_subset[categorical_columns])

# Applying one-hot encoding to categorical variables
encoder = OneHotEncoder(sparse=False, drop='first')
encoded_categorical = encoder.fit_transform(data_subset[categorical_columns])
encoded_categorical_df = pd.DataFrame(encoded_categorical, columns=encoder.get_feature_names_out(categorical_columns))
```

```
# Combining the encoded categorical variables with the numerical variables and dependent variables
final_data = pd.concat([
    encoded_categorical_df,
    data_subset[numerical_columns + dependent_variables].reset_index(drop=True)
], axis=1)

# Function to train and evaluate a Random Forest model
def train_evaluate_random_forest(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    return mse, r2

# Preparing the feature matrix X and training models for each dependent variable
X = final_data.drop(columns=dependent_variables)
performance_metrics_rf = {}
for dependent_variable in dependent_variables:
    y = final_data[dependent_variable]
    mse, r2 = train_evaluate_random_forest(X, y)
    performance_metrics_rf[dependent_variable] = {'MSE': mse, 'R2': r2}

# Displaying the performance metrics for Random Forest models
for variable, metrics in performance_metrics_rf.items():
    print(f"Dependent Variable: {variable} - Random Forest")
    print(f"Mean Squared Error: {metrics['MSE']}")
    print(f"R-squared: {metrics['R2']}\n")
```