DIC_Project_Phase_1_50608504

October 8, 2024

1 Phase 1

1.1 1: Problem Statement

1.1.1 1.1.1 Problem Statement

This project's goal is to make a detailed analysis on why road accidents are occurring a lot and in what trend, will be helpful in improving road safety measures & make the policy options which can reduce the number of accidents. This research would help to know the impacts of accident severity on the driver attributes, vehicle conditions, surface conditions and environmental conditions.

1.1.2 1.1.2 Potential Contribution & Importance

Road accidents pose a threat to health globally by resulting in significant fatalities and injuries of individuals worldwide. This evaluation plays a role in finding factors that play a vital role in accident prevention. The evidence of this review may support the implementation of measures of safety, improvement of driver education programs, and modification of road systems that can reduce accidents and save lives.

1.2 2: Ask Questions

1.2.1 Bhuvan Thirwani:

Question 1:

Question 2:

Harshit Malpani: 50608809

Question 1: What vehicles should the authorities focus more on to reduce the cases of road accidents and the severity of the road accidents

Question 2: Does the service period of the vehicle and ownership of the vehicle have any correlation with the accidents

Piyush Gulhane:

Question 1: What is the Impact of area, type of road cross-section, type of roads and road alignment on different types of Accidents

- This analysis will help us identify accident prone areas, common mistakes in road infrastructure, alignment and help us identify potential dark spots. It will help in future planning for Roads construction to avoid such road engineering mistakes like installing traffic signals, gradient of road, signboards, etc.
- Many a times slope of road, busy cross sections and other factors has influence on the accident, to identify and rectify these factors help in reduction of accidents.

Question 2: What is the impact of Environmental factors, Light(visibility) impact, Road surface, time of the day, etc. * This analysis will help us understand conditions/situations which forced human error, Most of time unavailability of light, less visibility, heavy rain can increase probability of accident. Appropriate changes in vehicle engineering and roads can help reduce casualties. * It is significant to identify conditions which affect driving experience.

[119]:

```

```

1.3 3: Data Retrieval

The dataset has been taken from KAGGLE. For this task, we have uploaded a copy of the dataset to a github repository and downloading the data from the github repository directly to the dataframe

[120]:

```
import pandas as pd

dataset = pd.read_csv('https://raw.githubusercontent.com/hmalpani/RTA-Dataset/
 ↪main/RTA_Dataset.csv')
```

[121]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12316 entries, 0 to 12315
Data columns (total 32 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Time                         12316 non-null  object
 1   Day_of_week                  12316 non-null  object
 2   Age_band_of_driver           12316 non-null  object
 3   Sex_of_driver                12316 non-null  object
 4   Educational_level            11575 non-null  object
 5   Vehicle_driver_relation      11737 non-null  object
 6   Driving_experience           11487 non-null  object
 7   Type_of_vehicle              11366 non-null  object
 8   Owner_of_vehicle             11834 non-null  object
 9   Service_year_of_vehicle      8388 non-null   object
 10  Defect_of_vehicle            7889 non-null   object
 11  Area_accident_occured        12077 non-null  object
 12  Lanes_or_Medians             11931 non-null  object
```

```
 13  Road_allignment             12174 non-null  object
 14  Types_of_Junction           11429 non-null  object
 15  Road_surface_type           12144 non-null  object
 16  Road_surface_conditions     12316 non-null  object
 17  Light_conditions            12316 non-null  object
 18  Weather_conditions          12316 non-null  object
 19  Type_of_collision           12161 non-null  object
 20  Number_of_vehicles_involved 12316 non-null  int64
 21  Number_of_casualties        12316 non-null  int64
 22  Vehicle_movement            12008 non-null  object
 23  Casualty_class              12316 non-null  object
 24  Sex_of_casualty             12316 non-null  object
 25  Age_band_of_casualty        12316 non-null  object
 26  Casualty_severity           12316 non-null  object
 27  Work_of_casuality           9118 non-null   object
 28  Fitness_of_casuality        9681 non-null   object
 29  Pedestrian_movement         12316 non-null  object
 30  Cause_of_accident           12316 non-null  object
 31  Accident_severity           12316 non-null  object
dtypes: int64(2), object(30)
memory usage: 3.0+ MB
```

[122]:
```
dataset.head()
```

[122]:
```
       Time Day_of_week Age_band_of_driver Sex_of_driver   Educational_level  \
0  17:02:00      Monday              18-30          Male   Above high school
1  17:02:00      Monday              31-50          Male  Junior high school
2  17:02:00      Monday              18-30          Male  Junior high school
3   1:06:00      Sunday              18-30          Male  Junior high school
4   1:06:00      Sunday              18-30          Male  Junior high school

  Vehicle_driver_relation Driving_experience      Type_of_vehicle  \
0                Employee              1-2yr           Automobile
1                Employee         Above 10yr  Public (> 45 seats)
2                Employee              1-2yr      Lorry (41?100Q)
3                Employee             5-10yr  Public (> 45 seats)
4                Employee              2-5yr                  NaN

  Owner_of_vehicle Service_year_of_vehicle  … Vehicle_movement  \
0            Owner              Above 10yr  …   Going straight
1            Owner                 5-10yrs  …   Going straight
2            Owner                     NaN  …   Going straight
3     Governmental                     NaN  …   Going straight
4            Owner                 5-10yrs  …   Going straight

   Casualty_class Sex_of_casualty Age_band_of_casualty Casualty_severity  \
0              na              na                   na                na
```

```
1               na              na                  na                 na
2  Driver or rider            Male               31-50                  3
3       Pedestrian          Female               18-30                  3
4               na              na                  na                 na

  Work_of_casuality Fitness_of_casuality Pedestrian_movement  \
0               NaN                  NaN    Not a Pedestrian
1               NaN                  NaN    Not a Pedestrian
2            Driver                  NaN    Not a Pedestrian
3            Driver               Normal    Not a Pedestrian
4               NaN                  NaN    Not a Pedestrian

            Cause_of_accident Accident_severity
0             Moving Backward     Slight Injury
1                  Overtaking     Slight Injury
2   Changing lane to the left    Serious Injury
3  Changing lane to the right     Slight Injury
4                  Overtaking     Slight Injury

[5 rows x 32 columns]
```

1.4 4: Data Cleaning

1.4.1 1) Remove Duplicate Values:

Removing duplicate values is an essential step of data cleaning for any data science project. It helps in reducing the bias where certain data points are represented multiple times. If the duplicate values are not removed, it can skew the results and therefore lead to incorrect conclusions

[123]:
```
# Remove duplicates
cleaned_dataset = dataset.drop_duplicates()
```

1.4.2 2) Validation

[124]:
```
# Remove entries with 'Number_of_vehicles_involved' = 0
cleaned_dataset =␣
↪cleaned_dataset[cleaned_dataset['Number_of_vehicles_involved'] != 0]
```

1.4.3 3) Detection and Removal of Outliers

[125]:
```
# code for outliers handling

numerical_columns = ['Number_of_vehicles_involved', 'Number_of_casualties']
for column in numerical_columns:
    if not pd.api.types.is_numeric_dtype(cleaned_dataset[column]):
        print(f"Column '{column}' should be numeric but contains non-numeric␣
↪data.")
```

```
def detect_outliers(column):
    Q1 = cleaned_dataset[column].quantile(0.05)
    Q3 = cleaned_dataset[column].quantile(0.95)
    IQR = Q3 - Q1
    outliers = cleaned_dataset[(cleaned_dataset[column] < (Q1 - 1.5 * IQR)) |␣
 ↪(cleaned_dataset[column] > (Q3 + 1.5 * IQR))]
    return outliers

for column in numerical_columns:
    outliers = detect_outliers(column)
    if not outliers.empty:
        print(f"Outliers detected in column '{column}':\n", outliers.shape)

def remove_outliers(df, column):
    Q1 = cleaned_dataset[column].quantile(0.05)
    Q3 = cleaned_dataset[column].quantile(0.95)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return cleaned_dataset[(cleaned_dataset[column] >= lower_bound) &␣
 ↪(cleaned_dataset[column] <= upper_bound)]

print("Shape before removing outliers:", cleaned_dataset.shape)
# Remove outliers from both columns
cleaned_dataset = remove_outliers(cleaned_dataset,␣
 ↪'Number_of_vehicles_involved')
cleaned_dataset = remove_outliers(cleaned_dataset, 'Number_of_casualties')

# Check the shape of the DataFrame after removal
print("Shape after removing outliers:", cleaned_dataset.shape)
```

```
Outliers detected in column 'Number_of_vehicles_involved':
 (7, 32)
Shape before removing outliers: (12316, 32)
Shape after removing outliers: (12309, 32)
```

1.4.4 4) Handling Missing Values:

In this step of Data Cleaning, we either remove or impute the missing values in the dataset

[126]:
```
# Find the number of missing values
missing_value_count = cleaned_dataset.isnull().sum()
missing_value_count
```

[126]:
```
Time                              0
Day_of_week                       0
Age_band_of_driver                0
Sex_of_driver                     0
```

```
Educational_level               741
Vehicle_driver_relation         579
Driving_experience              829
Type_of_vehicle                 950
Owner_of_vehicle                482
Service_year_of_vehicle        3923
Defect_of_vehicle              4427
Area_accident_occured           239
Lanes_or_Medians                385
Road_allignment                 142
Types_of_Junction               887
Road_surface_type               172
Road_surface_conditions           0
Light_conditions                  0
Weather_conditions                0
Type_of_collision               155
Number_of_vehicles_involved       0
Number_of_casualties              0
Vehicle_movement                306
Casualty_class                    0
Sex_of_casualty                   0
Age_band_of_casualty              0
Casualty_severity                 0
Work_of_casuality              3197
Fitness_of_casuality           2634
Pedestrian_movement               0
Cause_of_accident                 0
Accident_severity                 0
dtype: int64
```

[127]:

```
dataset_columns = cleaned_dataset.columns.tolist()
missing_values_columns = missing_value_count[missing_value_count > 0].index.
 ↪tolist()
print(missing_values_columns)
```

```
['Educational_level', 'Vehicle_driver_relation', 'Driving_experience',
'Type_of_vehicle', 'Owner_of_vehicle', 'Service_year_of_vehicle',
'Defect_of_vehicle', 'Area_accident_occured', 'Lanes_or_Medians',
'Road_allignment', 'Types_of_Junction', 'Road_surface_type',
'Type_of_collision', 'Vehicle_movement', 'Work_of_casuality',
'Fitness_of_casuality']
```

[128]:

```
# Replace missing values
cleaned_dataset['Educational_level'].
 ↪fillna(cleaned_dataset['Educational_level'].mode()[0], inplace=True)
cleaned_dataset['Vehicle_driver_relation'].fillna('Unknown', inplace=True)
```

```
cleaned_dataset['Driving_experience'].
 ↪fillna(cleaned_dataset['Driving_experience'].mode()[0], inplace=True)
cleaned_dataset['Type_of_vehicle'].fillna('Unknown', inplace=True)
cleaned_dataset['Owner_of_vehicle'].fillna('Unknown', inplace=True)
cleaned_dataset['Service_year_of_vehicle'].fillna('Unknown', inplace=True)
cleaned_dataset['Defect_of_vehicle'].fillna('No defect', inplace=True)
cleaned_dataset['Area_accident_occured'].fillna('Unknown', inplace=True)
cleaned_dataset['Lanes_or_Medians'].fillna('Unknown', inplace=True)
cleaned_dataset['Road_allignment'].fillna('Unknown', inplace=True)
cleaned_dataset['Types_of_Junction'].fillna('Unknown', inplace=True)
cleaned_dataset['Road_surface_type'].fillna('Unknown', inplace=True)
cleaned_dataset['Type_of_collision'].fillna('Unknown', inplace=True)
cleaned_dataset['Vehicle_movement'].fillna('Unknown', inplace=True)
cleaned_dataset['Work_of_casuality'].fillna('Unknown', inplace=True)
cleaned_dataset['Fitness_of_casuality'].fillna('Unknown', inplace=True)
```

```
<ipython-input-128-5dd3a9061be5>:2: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Educational_level'].fillna(cleaned_dataset['Educational_level
'].mode()[0], inplace=True)
<ipython-input-128-5dd3a9061be5>:3: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Vehicle_driver_relation'].fillna('Unknown', inplace=True)
<ipython-input-128-5dd3a9061be5>:4: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
```

```
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Driving_experience'].fillna(cleaned_dataset['Driving_experien
ce'].mode()[0], inplace=True)
<ipython-input-128-5dd3a9061be5>:5: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Type_of_vehicle'].fillna('Unknown', inplace=True)
<ipython-input-128-5dd3a9061be5>:6: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Owner_of_vehicle'].fillna('Unknown', inplace=True)
<ipython-input-128-5dd3a9061be5>:7: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Service_year_of_vehicle'].fillna('Unknown', inplace=True)
```

```
<ipython-input-128-5dd3a9061be5>:8: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Defect_of_vehicle'].fillna('No defect', inplace=True)
<ipython-input-128-5dd3a9061be5>:9: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Area_accident_occured'].fillna('Unknown', inplace=True)
<ipython-input-128-5dd3a9061be5>:10: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Lanes_or_Medians'].fillna('Unknown', inplace=True)
<ipython-input-128-5dd3a9061be5>:11: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
```

```
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Road_allignment'].fillna('Unknown', inplace=True)
<ipython-input-128-5dd3a9061be5>:12: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Types_of_Junction'].fillna('Unknown', inplace=True)
<ipython-input-128-5dd3a9061be5>:13: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Road_surface_type'].fillna('Unknown', inplace=True)
<ipython-input-128-5dd3a9061be5>:14: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Type_of_collision'].fillna('Unknown', inplace=True)
<ipython-input-128-5dd3a9061be5>:15: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
```

```
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Vehicle_movement'].fillna('Unknown', inplace=True)
<ipython-input-128-5dd3a9061be5>:16: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Work_of_casuality'].fillna('Unknown', inplace=True)
<ipython-input-128-5dd3a9061be5>:17: FutureWarning: A value is trying to be set
on a copy of a DataFrame or Series through chained assignment using an inplace
method.
The behavior will change in pandas 3.0. This inplace method will never work
because the intermediate object on which we are setting values always behaves as
a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using
'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value)
instead, to perform the operation inplace on the original object.


  cleaned_dataset['Fitness_of_casuality'].fillna('Unknown', inplace=True)
```

1.4.5 5) Correcting Errors:

In this data cleaning, we identify and fix the errors or incosistencies present in the data

[129]:
```
cleaned_dataset['Type_of_vehicle'] = cleaned_dataset['Type_of_vehicle'].
 ↪replace('Lorry (41?100Q)', 'Lorry (41 - 100 Q)')
cleaned_dataset['Type_of_vehicle'] = cleaned_dataset['Type_of_vehicle'].
 ↪replace('Lorry (11?40Q)', 'Lorry (11 - 40 Q)')
cleaned_dataset['Type_of_vehicle'] = cleaned_dataset['Type_of_vehicle'].
 ↪replace('Public (13?45 seats)', 'Public (13 - 45 seats)')
cleaned_dataset['Area_accident_occured'] =␣
 ↪cleaned_dataset['Area_accident_occured'].replace('  Recreational areas',␣
 ↪'Recreational areas')
```

```
cleaned_dataset['Area_accident_occured'] =␣
↪cleaned_dataset['Area_accident_occured'].replace('  Market areas', 'Market␣
↪areas')
cleaned_dataset['Area_accident_occured'] =␣
↪cleaned_dataset['Area_accident_occured'].replace(' Church areas', 'Church␣
↪areas')
cleaned_dataset['Area_accident_occured'] =␣
↪cleaned_dataset['Area_accident_occured'].replace(' Hospital areas',␣
↪'Hospital areas')
cleaned_dataset['Area_accident_occured'] =␣
↪cleaned_dataset['Area_accident_occured'].replace(' Industrial areas',␣
↪'Industrial areas')
cleaned_dataset['Area_accident_occured'] =␣
↪cleaned_dataset['Area_accident_occured'].replace(' Outside rural areas',␣
↪'Outside rural areas')
cleaned_dataset['Area_accident_occured'] =␣
↪cleaned_dataset['Area_accident_occured'].replace('Rural village areasOffice␣
↪areas', 'Rural Office areas')
cleaned_dataset['Road_allignment'] = cleaned_dataset['Road_allignment'].
↪replace('Tangent road with mountainous terrain and', 'Tangent road with␣
↪mountainous terrain')
cleaned_dataset['Fitness_of_casuality'] =␣
↪cleaned_dataset['Fitness_of_casuality'].replace('NormalNormal', 'Normal')
cleaned_dataset['Casualty_severity'] = cleaned_dataset['Casualty_severity'].
↪replace('na', 'Unknown')
```

1.4.6 6) Standardize the Data

a) Convert all the entries in `Time` column to a consistent format.
b) Convert `Over 51` to `51 and Over` in the `Age_band_of_driver` column

[130]:
```
# Standardize the 'Time' column
cleaned_dataset['Time'] = pd.to_datetime(cleaned_dataset['Time'], format='%H:%M:
↪%S').dt.time
# Make 'Over 51' to '51 and Over' for Driver Age band
cleaned_dataset['Age_band_of_driver'] = cleaned_dataset['Age_band_of_driver'].
↪replace('Over 51', '51 and Over')
```

1.4.7 7) Parsing the data

Convert all the text in the dataset to lowercase to ensure consistency. This helps in avoiding the situations where same words with different cases are considered different

[131]:
```
# Make all the characters to lowercase
cleaned_dataset = cleaned_dataset.map(lambda x: x.lower() if isinstance(x, str)␣
↪else x)
```

1.4.8 8) Feature Engineering

[132]:
```
print(cleaned_dataset['Time'].head())
cleaned_dataset['Hour'] = pd.to_datetime(cleaned_dataset['Time'], format='%H:%M:
↪%S').dt.hour
Time_of_dat = ['Night', 'Morning', 'Noon', 'Evening']

def categorize_time_of_day(hour):
    if 5 <= hour < 12:
        return 2
    elif 12 <= hour < 17:
        return 3
    elif 17 <= hour < 21:
        return 4
    else:
        return 1


cleaned_dataset['Time_of_day'] = cleaned_dataset['Hour'].
↪apply(categorize_time_of_day)

print("Data head after categorizing and encoding Time_of_day:\n")
cleaned_dataset[['Time', 'Hour', 'Time_of_day']].head()
```

```
0    17:02:00
1    17:02:00
2    17:02:00
3    01:06:00
4    01:06:00
Name: Time, dtype: object
Data head after categorizing and encoding Time_of_day:
```

[132]:

	Time	Hour	Time_of_day
0	17:02:00	17	4
1	17:02:00	17	4
2	17:02:00	17	4
3	01:06:00	1	1
4	01:06:00	1	1

9) Ordinal & One Hot Encoding

[133]:
```
from sklearn.preprocessing import OneHotEncoder

encoding_dict = {
    'Day_of_week': 'ordinal',
    'Age_band_of_driver': 'ordinal',
    'Sex_of_driver': 'one_hot',
```

```
    'Educational_level': 'ordinal',
    'Vehicle_driver_relation': 'one_hot',
    'Driving_experience': 'ordinal',
    'Type_of_vehicle': 'one_hot',
    'Owner_of_vehicle': 'one_hot',
    'Service_year_of_vehicle': 'ordinal',
    'Defect_of_vehicle': 'one_hot',
    'Area_accident_occured': 'one_hot',
    'Lanes_or_Medians': 'one_hot',
    'Road_allignment': 'one_hot',
    'Types_of_Junction': 'one_hot',
    'Road_surface_type': 'one_hot',
    'Road_surface_conditions': 'ordinal',
    'Light_conditions': 'one_hot',
    'Weather_conditions': 'one_hot',
    'Type_of_collision': 'one_hot',
    'Vehicle_movement': 'one_hot',
    'Casualty_class': 'one_hot',
    'Sex_of_casualty': 'one_hot',
    'Age_band_of_casualty': 'ordinal',
    'Casualty_severity': 'ordinal',
    'Work_of_casuality': 'one_hot',
    'Fitness_of_casuality': 'one_hot',
    'Pedestrian_movement': 'one_hot',
    'Cause_of_accident': 'one_hot',
    'Accident_severity': 'ordinal'
}

ordinal_mappings = {
    'Day_of_week': {
        'Monday': 0, 'Tuesday': 1, 'Wednesday': 2, 'Thursday': 3,
        'Friday': 4, 'Saturday': 5, 'Sunday': 6, 'Unknown': -1
    },
    'Age_band_of_driver': {
        'Under 18': 0, '18-30': 1, '31-50': 2, 'Over 51': 3, 'Unknown': -1
    },
    'Educational_level': {
        'Illiterate': 0, 'Writing & reading': 1, 'Elementary school': 2,
        'Junior high school': 3, 'High school': 4, 'Above high school': 5,
        'Unknown': -1
    },
    'Driving_experience': {
        'No Licence': 0, 'Below 1yr': 1, '1-2yr': 2, '2-5yr': 3, '5-10yr': 4,
        'Above 10yr': 5, 'unknown': -1
    },
    'Service_year_of_vehicle': {
        'Below 1yr': 0, '1-2yr': 1, '2-5yrs': 2, '5-10yrs': 3,
```

```
        'Above 10yr': 4, 'Unknown': -1
    },
    'Road_surface_conditions': {
        'Dry': 0, 'Wet or damp': 1, 'Snow': 2, 'Flood over 3cm. deep': 3,␣
↪'Unknown': -1
    },
    'Age_band_of_casualty': {
        'Under 18': 0, '18-30': 1, '31-50': 2, 'Over 51': 3, '5': 4, 'na': -1,␣
↪'Unknown': -1
    },
    'Casualty_severity': {
        '3': 0, '2': 1, '1': 2, 'na': -1, 'Unknown': -1
    },
    'Accident_severity': {
        'Slight Injury': 0, 'Serious Injury': 1, 'Fatal injury': 2, 'Unknown':␣
↪-1
    }
}

def apply_encoding(df, encoding_dict, ordinal_mappings):
    one_hot_encoder = OneHotEncoder(sparse_output=False, drop='first')

    for column, encoding_type in encoding_dict.items():
        if encoding_type == 'ordinal':
            # Apply ordinal encoding using a mapping dictionary
            if column in ordinal_mappings:
                df[f"{column}_ordinal"] = df[column].
↪map(ordinal_mappings[column])
            else:
                print(f"No ordinal mapping provided for column: {column}")

        elif encoding_type == 'one_hot':
            one_hot_encoded_df = pd.get_dummies(df[column], prefix=column,␣
↪drop_first=True)
            df = pd.concat([df, one_hot_encoded_df], axis=1)

        else:
            print(f"Unknown encoding type: {encoding_type} for column:␣
↪{column}")

    return df

cleaned_dataset = apply_encoding(cleaned_dataset, encoding_dict,␣
↪ordinal_mappings)

cleaned_dataset.head()
```

[133]:

```
       Time Day_of_week Age_band_of_driver Sex_of_driver   Educational_level  \
0  17:02:00      monday              18-30          male   above high school
1  17:02:00      monday              31-50          male  junior high school
2  17:02:00      monday              18-30          male  junior high school
3  01:06:00      sunday              18-30          male  junior high school
4  01:06:00      sunday              18-30          male  junior high school

  Vehicle_driver_relation Driving_experience      Type_of_vehicle  \
0                employee              1-2yr           automobile
1                employee         above 10yr  public (> 45 seats)
2                employee              1-2yr   lorry (41 - 100 q)
3                employee             5-10yr  public (> 45 seats)
4                employee              2-5yr              unknown

  Owner_of_vehicle Service_year_of_vehicle  …  \
0            owner              above 10yr  …
1            owner                 5-10yrs  …
2            owner                 unknown  …
3     governmental                 unknown  …
4            owner                 5-10yrs  …

  Cause_of_accident_no priority to pedestrian  \
0                                       False
1                                       False
2                                       False
3                                       False
4                                       False

  Cause_of_accident_no priority to vehicle Cause_of_accident_other  \
0                                    False                   False
1                                    False                   False
2                                    False                   False
3                                    False                   False
4                                    False                   False

  Cause_of_accident_overloading Cause_of_accident_overspeed  \
0                         False                       False
1                         False                       False
2                         False                       False
3                         False                       False
4                         False                       False

  Cause_of_accident_overtaking Cause_of_accident_overturning  \
0                        False                         False
1                         True                         False
2                        False                         False
3                        False                         False
```

```
4                        True                        False

  Cause_of_accident_turnover Cause_of_accident_unknown  \
0                      False                     False
1                      False                     False
2                      False                     False
3                      False                     False
4                      False                     False

  Accident_severity_ordinal
0                       NaN
1                       NaN
2                       NaN
3                       NaN
4                       NaN

[5 rows x 184 columns]
```

1.5 5: Exploratory Data Analysis (EDA)

1.5.1 Bhuvan Thirwani:

Question 1:

Question 2:

1.5.2 Harshit Malpani: 50608809

Question 1:

Question 2:

1.5.3 Piyush Gulhane:

Question 1:

Analysis of impact of Roads,type of Road cross-section, type of Roads and Road alignment on different types of Accidents

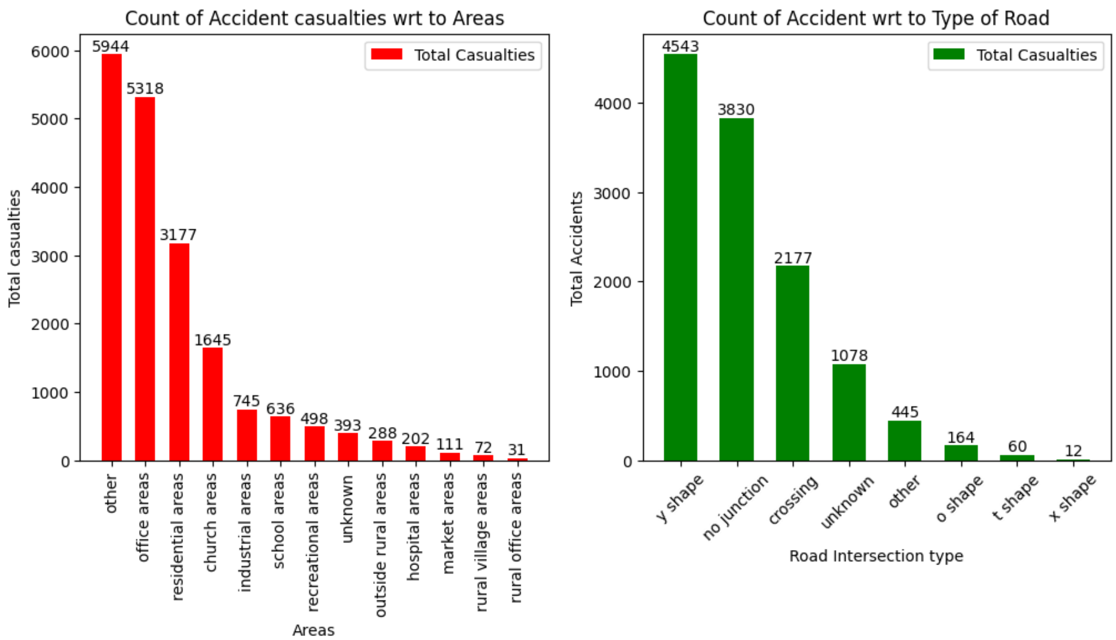
Hypothsis 1: Imapct of Area on Accidents and its correlation with other factors like lanes, cross-section, driving error etc. * The Below graphs helps us identify areas with higher number of accident casualties showing us to focus on these areas to improve road infrastructure. It shows the heavy movement of traffic in Urban Office areas and Residential areas. Followed by Church and Industrial area

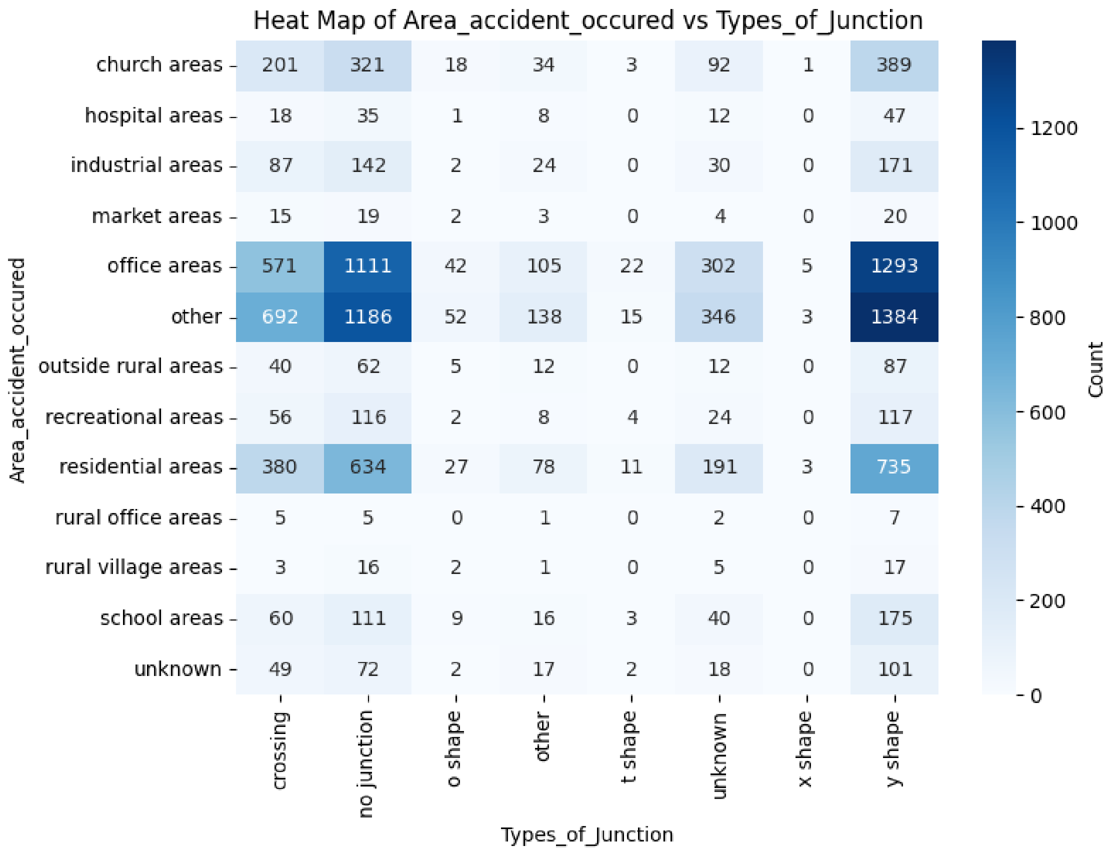
- The graph 'Count of Accident casualties wrt to Type of Road' shows the accidents that occur at road intersections. we can see that Y shape intersections are prone to accidents as compared to t-shape,X-shape,O-shape crossing. Suggesting to avoid Y shape crossing where-ever possible.

- Heat Map 'Area_accident_occured vs *Types_of_Junction*' shows the number of accidents in an area at junction of roads. We see that accidents at y-shape junctions are higher in School-Church-Residentail and Industrial areas with office & other areas topping the chart.
- Significant number of accident are seenon Road Crossing as well
- O-shape, T-shape junctions account for small amount of accidents.

[134]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12,5))
df=cleaned_dataset.
 ↪groupby("Area_accident_occured",as_index=False)['Number_of_casualties'].
 ↪sum().sort_values(by="Number_of_casualties",ascending=False)
df1=cleaned_dataset.
 ↪groupby("Types_of_Junction",as_index=False)['Number_of_casualties'].count().
 ↪sort_values(by="Number_of_casualties",ascending=False)
df.head(10)
bar_width = 0.6
plt.subplot(1, 2, 1)
hdi_bar =plt.bar(df["Area_accident_occured"], df["Number_of_casualties"],␣
 ↪width=bar_width, label='Total Casualties', color='red', align='center')
for bar in hdi_bar:
    death_cnt = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, death_cnt, str(death_cnt),
             ha='center', va='bottom')
plt.legend(loc='upper right')
plt.xlabel('Areas')
plt.xticks(rotation=90)
plt.ylabel('Total casualties')
plt.title('Count of Accident casualties wrt to Areas')
plt.legend()
plt.subplot(1, 2, 2)
hdi_bar =plt.bar(df1["Types_of_Junction"], df1["Number_of_casualties"],␣
 ↪width=bar_width, label='Total Casualties', color='green', align='center')
for bar in hdi_bar:
    death_cnt = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, death_cnt, str(death_cnt),
             ha='center', va='bottom')
plt.legend(loc='upper right')
plt.xlabel('Road Intersection type')
plt.xticks(rotation=45)
plt.ylabel('Total Accidents')
plt.title('Count of Accident wrt to Type of Road')
plt.legend()
plt.show()
```

```
# Heat Map of Area_accident_occured vs Types_of_Junction
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Area_accident_occured"],
  columns=df["Types_of_Junction"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Blues',
  cbar_kws={'label': 'Count'})
plt.title('Heat Map of Area_accident_occured vs Types_of_Junction')
plt.xlabel('Types_of_Junction')
plt.ylabel('Area_accident_occured')
plt.show()
```



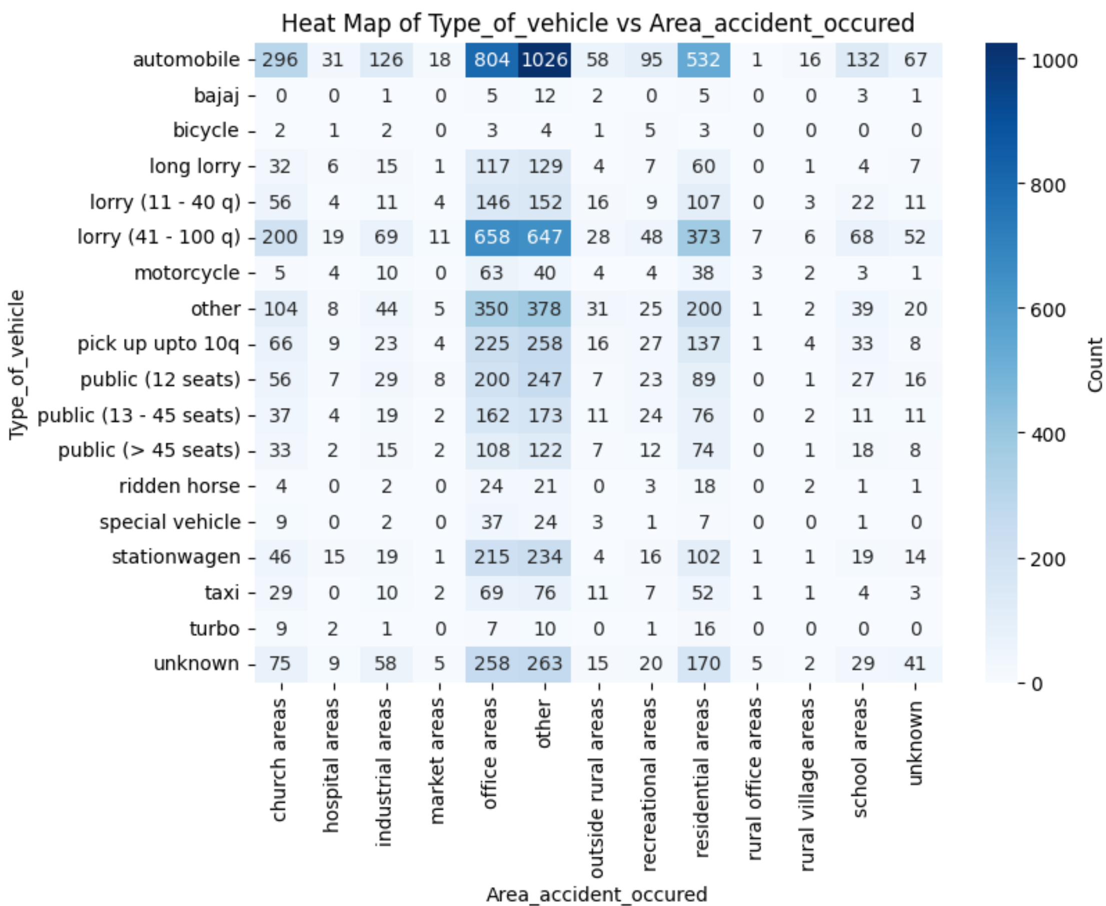


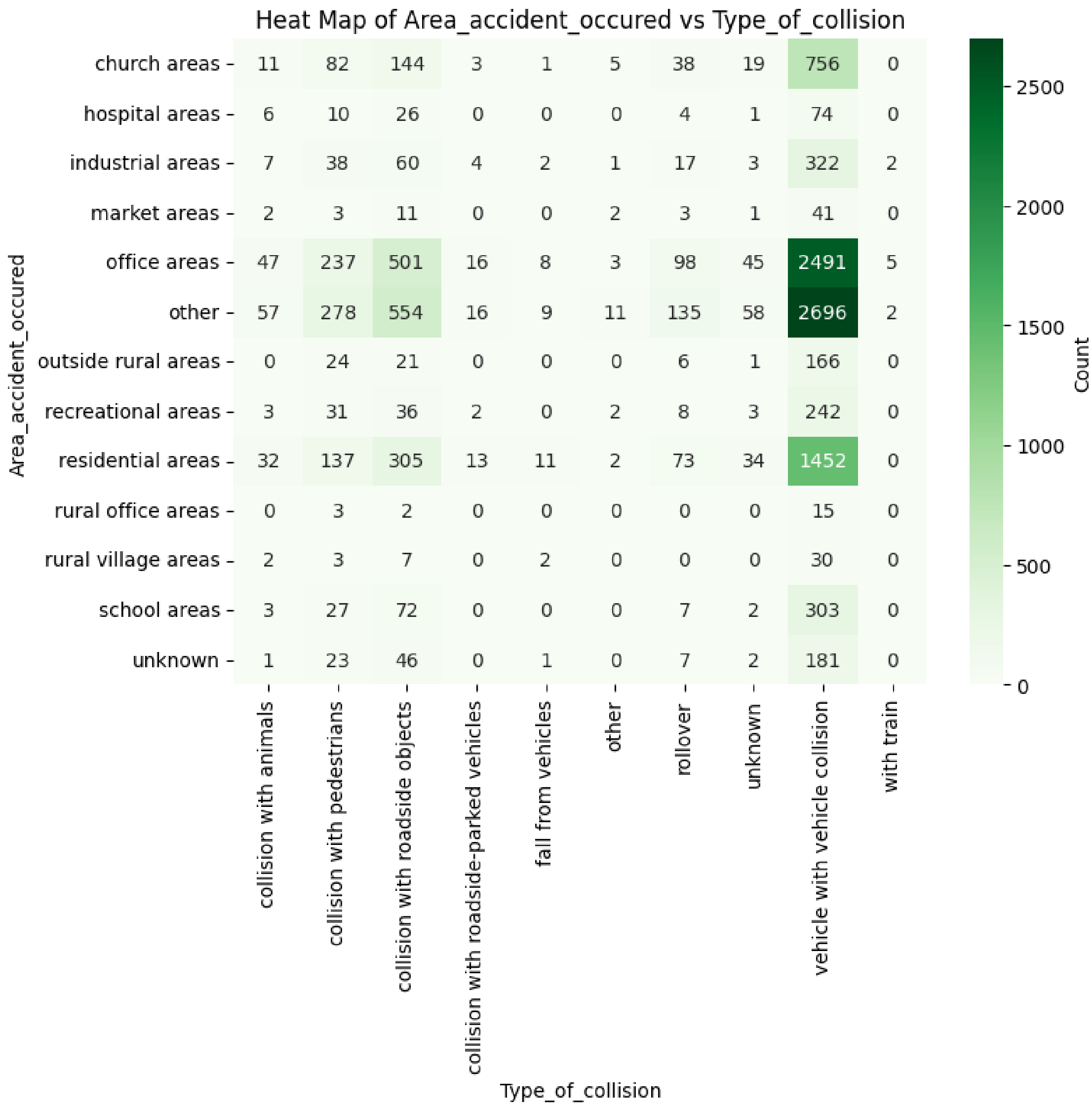
	crossing	no junction	o shape	other	t shape	unknown	x shape	y shape
church areas	201	321	18	34	3	92	1	389
hospital areas	18	35	1	8	0	12	0	47
industrial areas	87	142	2	24	0	30	0	171
market areas	15	19	2	3	0	4	0	20
office areas	571	1111	42	105	22	302	5	1293
other	692	1186	52	138	15	346	3	1384
outside rural areas	40	62	5	12	0	12	0	87
recreational areas	56	116	2	8	4	24	0	117
residential areas	380	634	27	78	11	191	3	735
rural office areas	5	5	0	1	0	2	0	7
rural village areas	3	16	2	1	0	5	0	17
school areas	60	111	9	16	3	40	0	175
unknown	49	72	2	17	2	18	0	101

- 'Type_of_vehicle vs Area_accident_occured' distribution helps us understand the movement of different type of vehicles in a area and adressing to the issues faced by them.
- From the map 'Area_accident_occured vs Type_of_collision' most common acident type is vehicle on vehicle collision, it can be seen across all the area.Followed by hitting roadside objects and pedestrians in office and residential area.
- Significant casualties are seen by Rollover in office area.

[135]:
```
# Heat Map of Type_of_vehicle vs Area_accident_occured
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Type_of_vehicle"],␣
↪columns=df["Area_accident_occured"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Blues',␣
↪cbar_kws={'label': 'Count'})
plt.title('Heat Map of Type_of_vehicle vs Area_accident_occured')
plt.xlabel('Area_accident_occured')
plt.ylabel('Type_of_vehicle')
plt.show()
```

```python
# Heat Map of Area_accident_occured vs Type_of_collision
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Area_accident_occured"],
    columns=df["Type_of_collision"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Greens',
    cbar_kws={'label': 'Count'})
plt.title('Heat Map of Area_accident_occured vs Type_of_collision')
plt.xlabel('Type_of_collision')
plt.ylabel('Area_accident_occured')
plt.show()
```





Hypothesis 2: Studying Road Alignments, Junctions, causes of accidents and their correlation. Lane Markings at accident location distribution * We can see that two way undivided roads and two way road divided with broken lines has higher percent of errors. This helps to understand driving patterns and designing road with proper medians

Heat Map of Cause_of_accident vs Lanes_or_Medians * On Double carriageway, we see accidents due to lane changing and overspeeding * For Two way roads with overtake allowed, high number of accidents are seen during overtaking and not giving priority to other vehicle or pedestrain

- High accidents are seen when no distance is kepet between vehicles and moving backwards on road. Measures should be taken to avoid such irresponsible driving

Heat Map of Road_allignment vs Types_of_Junction

- Apart from flat teraain, combination of steep downward grade on mountainous terrain with y-Shape junction sees 161 accidents, similar with tangent roads on moutainous terrain.

Heat Map of Lanes_or_Medians vs Types_of_Junction

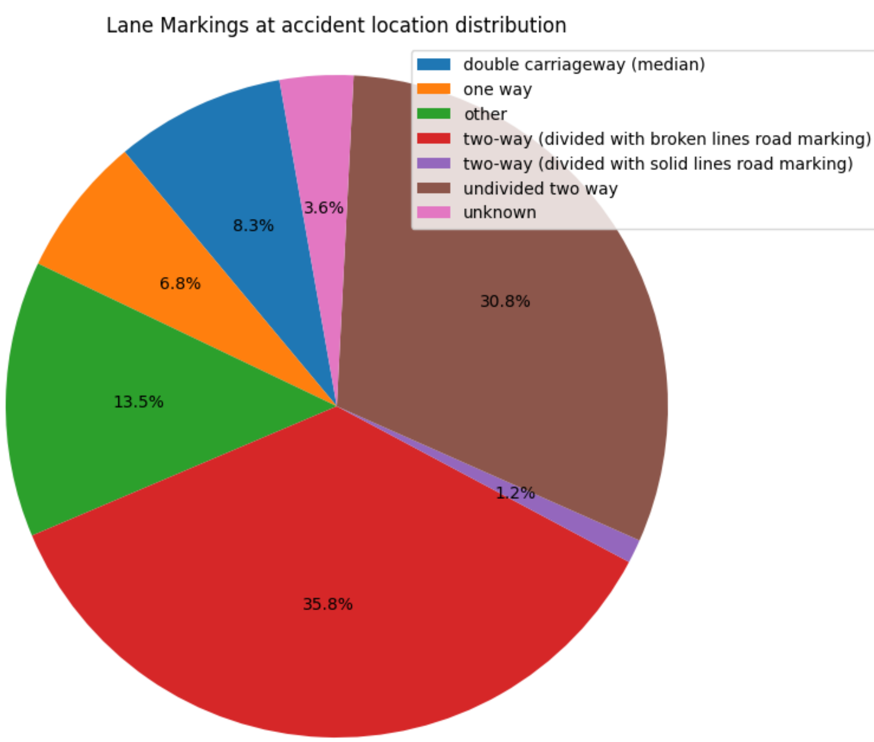
- This map helps us understand the junctions and lane structure at accident spot,one way combination with Y-shape,crossing,no junction have higher accident rates.

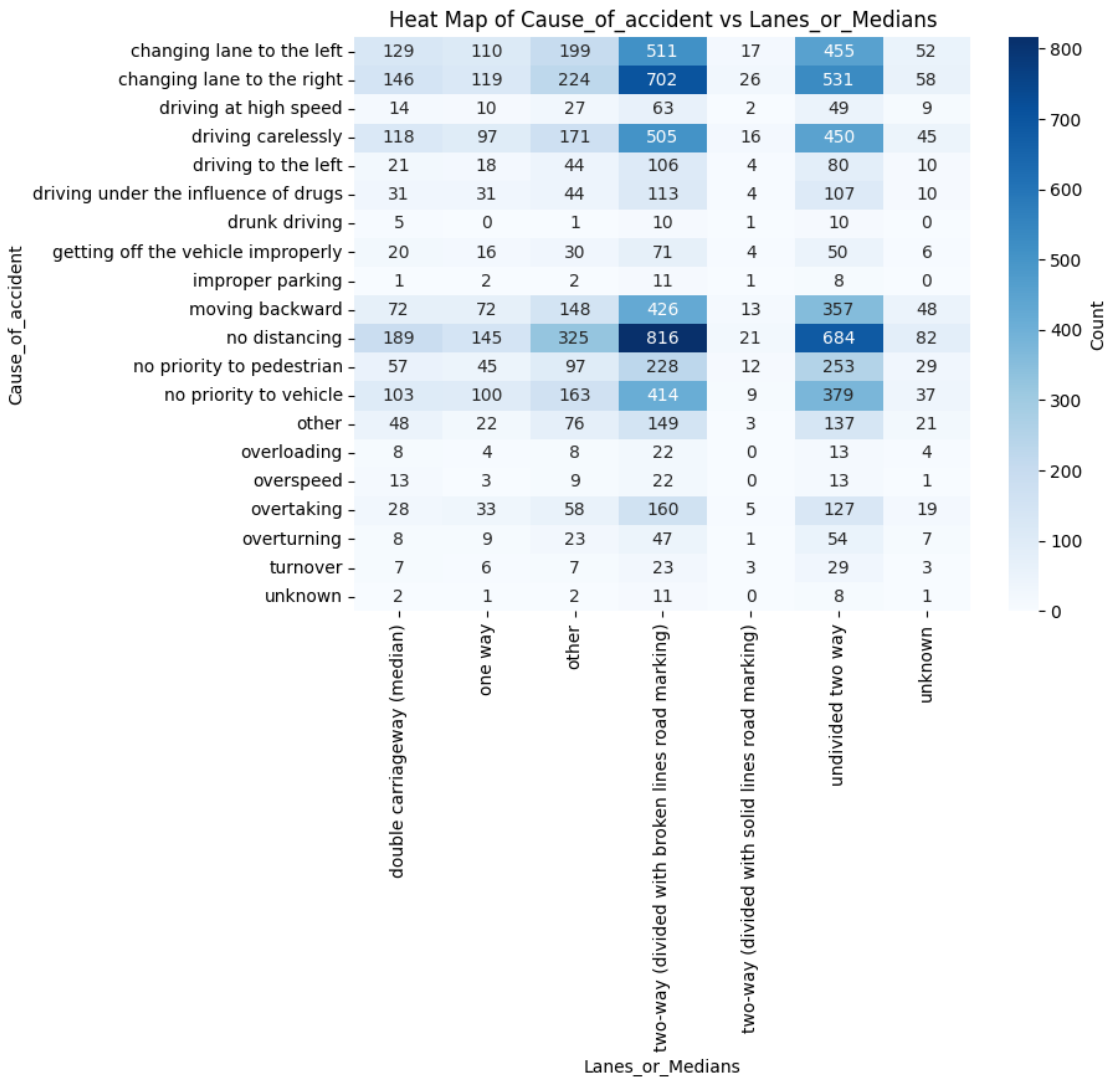
Heat Map of Vehicle_movement vs Type_of_collision

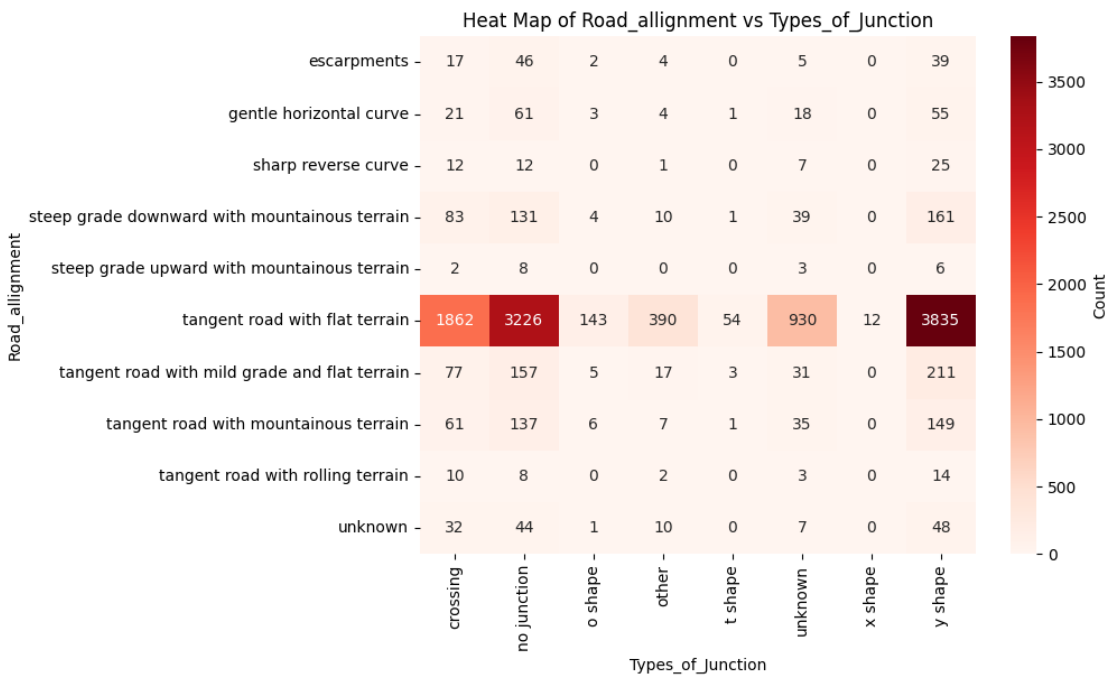
- For highest category of collisions i.e. vehicle to vehicle collisions we see the actual reason and count of collision from above heat map.
- Moving backward, reversing , turnover account, Entering junction for accident along with straight on hitting

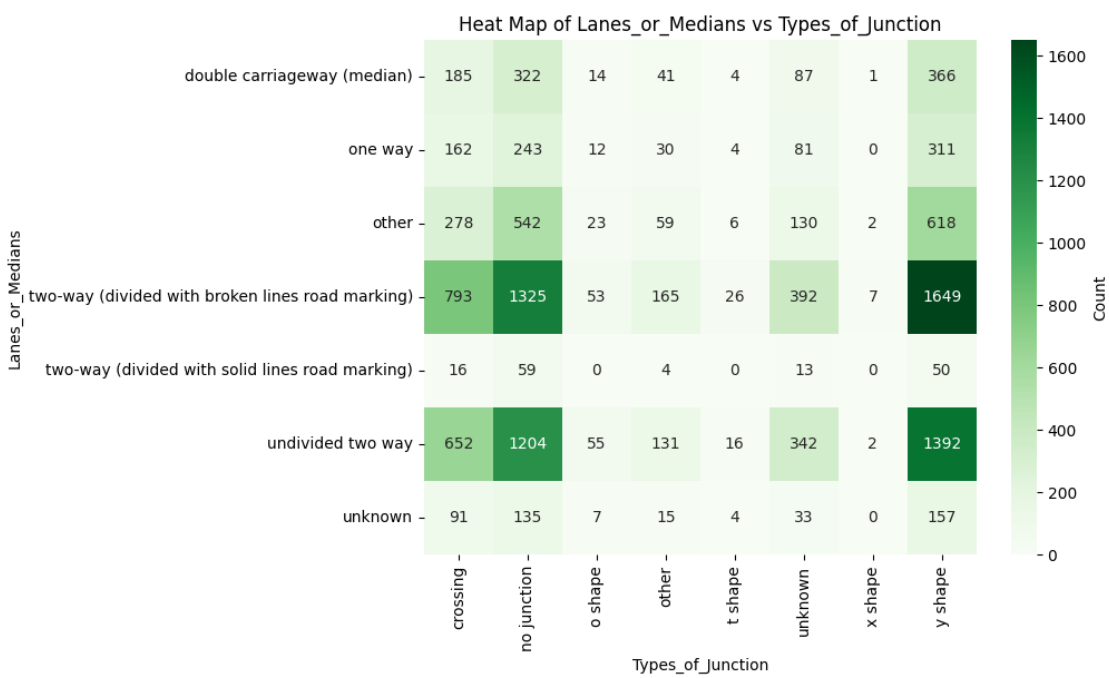
[136]:
```
# Lane Markings at accident location distribution
df=cleaned_dataset.
 ↪groupby("Lanes_or_Medians",as_index=False)["Number_of_casualties"].count()
plt.figure(figsize=(12, 8))
plt.pie(df['Number_of_casualties'], autopct='%1.1f%%', startangle=100)
plt.title('Lane Markings at accident location distribution')
plt.legend(df['Lanes_or_Medians'], loc='upper right')
plt.axis('equal')
plt.show()

# Heat Map of Cause_of_accident vs Lanes_or_Medians
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Cause_of_accident"],␣
 ↪columns=df["Lanes_or_Medians"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Blues',␣
 ↪cbar_kws={'label': 'Count'})
plt.title('Heat Map of Cause_of_accident vs Lanes_or_Medians')
plt.xlabel('Lanes_or_Medians')
plt.ylabel('Cause_of_accident')
plt.show()

# Heat Map of Road_allignment vs Types_of_Junction
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Road_allignment"],␣
 ↪columns=df["Types_of_Junction"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Reds',␣
 ↪cbar_kws={'label': 'Count'})
plt.title('Heat Map of Road_allignment vs Types_of_Junction')
plt.xlabel('Types_of_Junction')
plt.ylabel('Road_allignment')
plt.show()

# Heat Map of Lanes_or_Medians vs Types_of_Junction
```

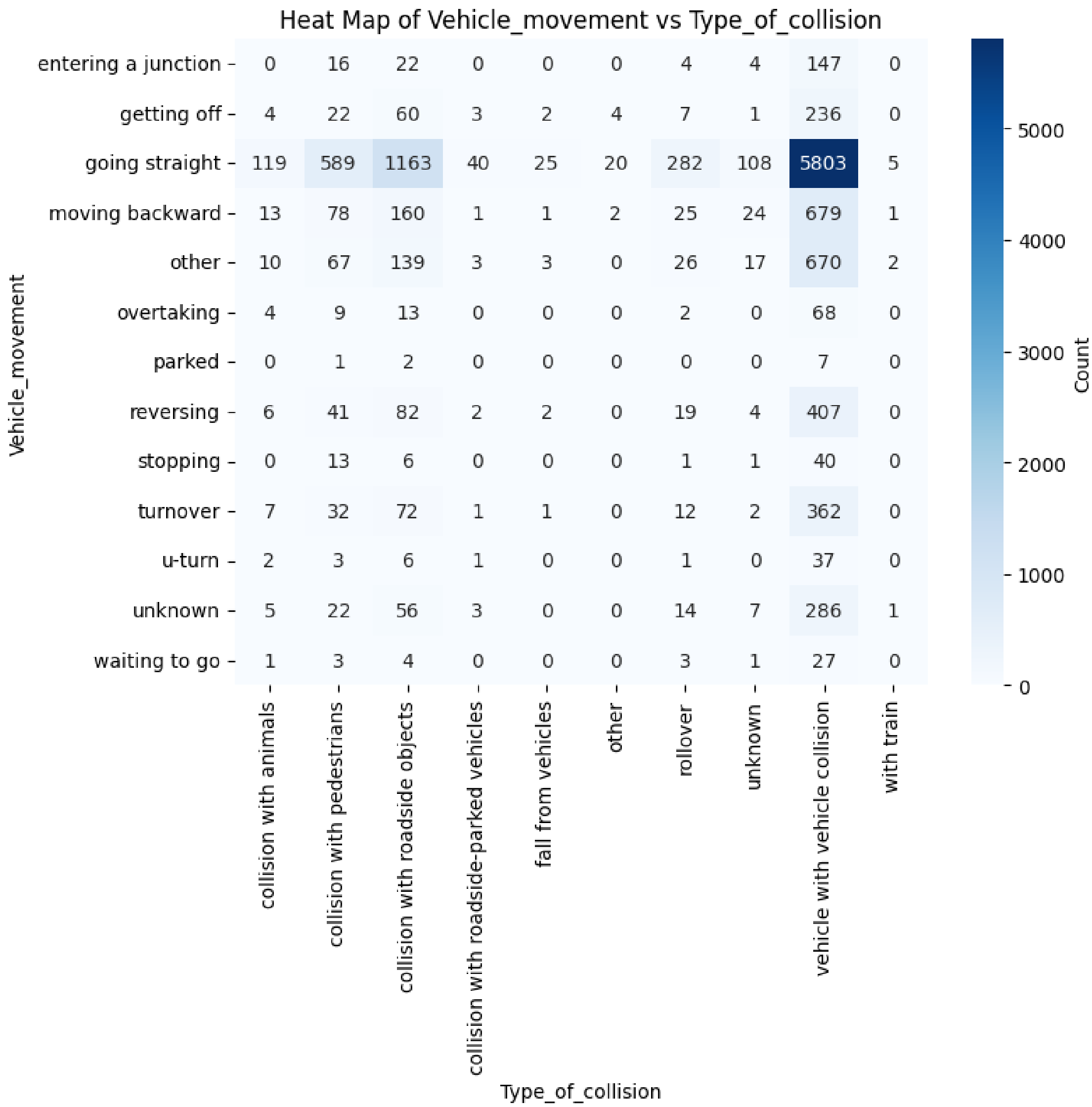
```python
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Lanes_or_Medians"],␣
↪columns=df["Types_of_Junction"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Greens',␣
↪cbar_kws={'label': 'Count'})
plt.title('Heat Map of Lanes_or_Medians vs Types_of_Junction')
plt.xlabel('Types_of_Junction')
plt.ylabel('Lanes_or_Medians')
plt.show()


# Heat Map of Vehicle_movement vs Type_of_collision
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Vehicle_movement"],␣
↪columns=df["Type_of_collision"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Blues',␣
↪cbar_kws={'label': 'Count'})
plt.title('Heat Map of Vehicle_movement vs Type_of_collision')
plt.xlabel('Type_of_collision')
plt.ylabel('Vehicle_movement')
plt.show()
```




Heat Map of Cause_of_accident vs Lanes_or_Medians
Cause_of_accident
Lanes_or_Medians
Count
changing lane to the left 129 110 199 511 17 455 52
changing lane to the right 146 119 224 702 26 531 58
driving at high speed 14 10 27 63 2 49 9
driving carelessly 118 97 171 505 16 450 45
driving to the left 21 18 44 106 4 80 10
driving under the influence of drugs 31 31 44 113 4 107 10
drunk driving 5 0 1 10 1 10 0
getting off the vehicle improperly 20 16 30 71 4 50 6
improper parking 1 2 2 11 1 8 0
moving backward 72 72 148 426 13 357 48
no distancing 189 145 325 816 21 684 82
no priority to pedestrian 57 45 97 228 12 253 29
no priority to vehicle 103 100 163 414 9 379 37
other 48 22 76 149 3 137 21
overloading 8 4 8 22 0 13 4
overspeed 13 3 9 22 0 13 1
overtaking 28 33 58 160 5 127 19
overturning 8 9 23 47 1 54 7
turnover 7 6 7 23 3 29 3
unknown 2 1 2 11 0 8 1
double carriageway (median)
one way
other
two-way (divided with broken lines road marking)
two-way (divided with solid lines road marking)
undivided two way
unknown
0
100
200
300
400
500
600
700
800


Heat Map of Road_allignment vs Types_of_Junction
Road_allignment
escarpments
gentle horizontal curve
sharp reverse curve
steep grade downward with mountainous terrain
steep grade upward with mountainous terrain
tangent road with flat terrain
tangent road with mild grade and flat terrain
tangent road with mountainous terrain
tangent road with rolling terrain
unknown
17 46 2 4 0 5 0 39
21 61 3 4 1 18 0 55
12 12 0 1 0 7 0 25
83 131 4 10 1 39 0 161
2 8 0 0 0 3 0 6
1862 3226 143 390 54 930 12 3835
77 157 5 17 3 31 0 211
61 137 6 7 1 35 0 149
10 8 0 2 0 3 0 14
32 44 1 10 0 7 0 48
crossing
no junction
o shape
other
t shape
unknown
x shape
y shape
Types_of_Junction
Count
0
500
1000
1500
2000
2500
3000
3500


Heat Map of Lanes_or_Medians vs Types_of_Junction
Lanes_or_Medians
double carriageway (median)
one way
other
two-way (divided with broken lines road marking)
two-way (divided with solid lines road marking)
undivided two way
unknown
185 322 14 41 4 87 1 366
162 243 12 30 4 81 0 311
278 542 23 59 6 130 2 618
793 1325 53 165 26 392 7 1649
16 59 0 4 0 13 0 50
652 1204 55 131 16 342 2 1392
91 135 7 15 4 33 0 157
crossing
no junction
o shape
other
t shape
unknown
x shape
y shape
Types_of_Junction
Count
0
200
400
600
800
1000
1200
1400
1600



	collision with animals	collision with pedestrians	collision with roadside objects	collision with roadside-parked vehicles	fall from vehicles	other	rollover	unknown	vehicle with vehicle collision	with train
entering a junction	0	16	22	0	0	0	4	4	147	0
getting off	4	22	60	3	2	4	7	1	236	0
going straight	119	589	1163	40	25	20	282	108	5803	5
moving backward	13	78	160	1	1	2	25	24	679	1
other	10	67	139	3	3	0	26	17	670	2
overtaking	4	9	13	0	0	0	2	0	68	0
parked	0	1	2	0	0	0	0	0	7	0
reversing	6	41	82	2	2	0	19	4	407	0
stopping	0	13	6	0	0	0	1	1	40	0
turnover	7	32	72	1	1	0	12	2	362	0
u-turn	2	3	6	1	0	0	1	0	37	0
unknown	5	22	56	3	0	0	14	7	286	1
waiting to go	1	3	4	0	0	0	3	1	27	0

Question 2:

Analysis of impact of Environmental factors, Light(visibility) impact, Road surface, time of the day, etc on driving skills

Hypothesis 1 : Analysing the Accident patterns over days of week and different times of the day. Count of Accident casualties wrt to Time of Day

- We see highest number of accidents in evening period which is same as end of office hours. Followed by Night time.
- Number of Early Morning and Midnight accidents are very less.

Count of Accident wrt Days of week * We see the count of accident per week day. We see on Weekends there is significant less count as compared to week days. * Office areas count for more accidents, significant from this data of weekdays.

Heat Map of Day_of_week vs Area_accident_occured

- Office area has less accident on weekends
- same trend is visible for Residential and Industrial areas indicating less activity
- For Church area we see high numbers especially for friday

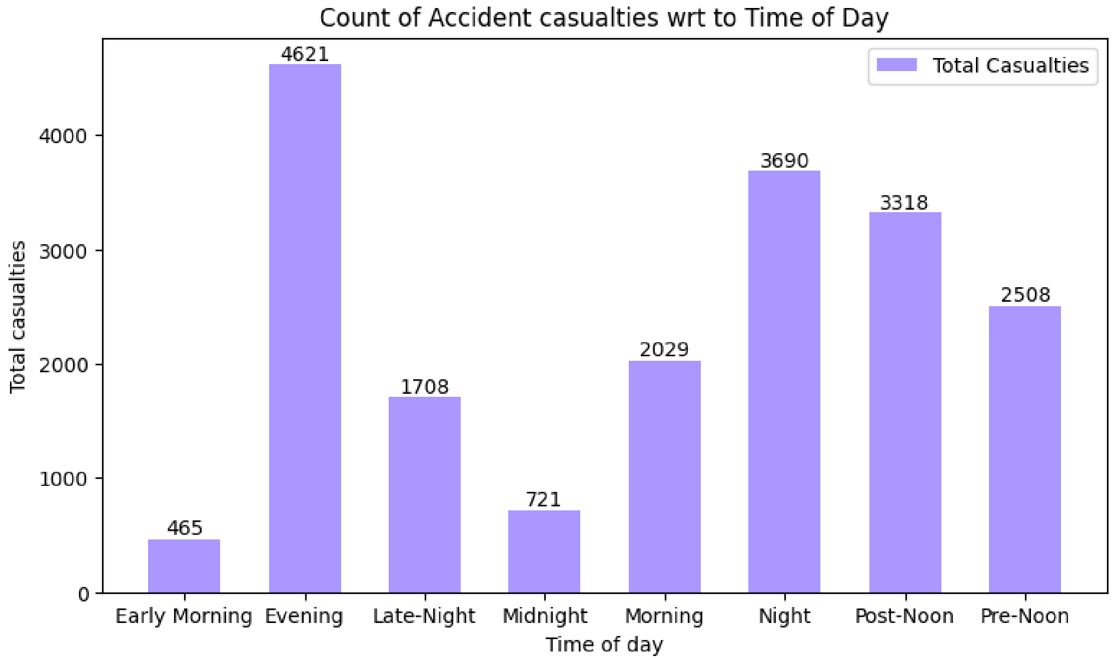
Road surface distribution

- From this we can say that most of the roads are made of Asphalt these days. Apart from that we can see earth roads and gravel roads as well

[137]:
```
def categorize_time_of_dayby3(hour):
    if 3<= hour < 6:
        return 'Early Morning'
    elif 6 <= hour < 9:
        return 'Morning'
    elif 9 <= hour < 12:
        return 'Pre-Noon'
    elif 12 <= hour < 15:
        return 'Post-Noon'
    elif 15 <= hour < 18:
        return 'Evening'
    elif 18 <= hour < 21:
        return 'Night'
    elif 21 <= hour < 24:
        return 'Late-Night'
    else:
        return 'Midnight'

df_new=cleaned_dataset
df_new['Time_of_day_3'] = df_new['Hour'].apply(categorize_time_of_dayby3)

plt.figure(figsize=(9,5))
df=df_new.groupby("Time_of_day_3",as_index=False)['Number_of_casualties'].sum()
#df1=cleaned_dataset.
 ↪groupby("Types_of_Junction",as_index=False)['Number_of_casualties'].count().
 ↪sort_values(by="Number_of_casualties",ascending=False)
df.head(10)
bar_width = 0.6
hdi_bar =plt.bar(df["Time_of_day_3"], df["Number_of_casualties"],␣
 ↪width=bar_width, label='Total Casualties', color='#AA97FF', align='center')
for bar in hdi_bar:
    death_cnt = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, death_cnt, str(death_cnt),
             ha='center', va='bottom')
plt.legend(loc='upper right')
plt.xlabel('Time of day')
plt.ylabel('Total casualties')
```

```
plt.title('Count of Accident casualties wrt to Time of Day')
plt.legend()
```

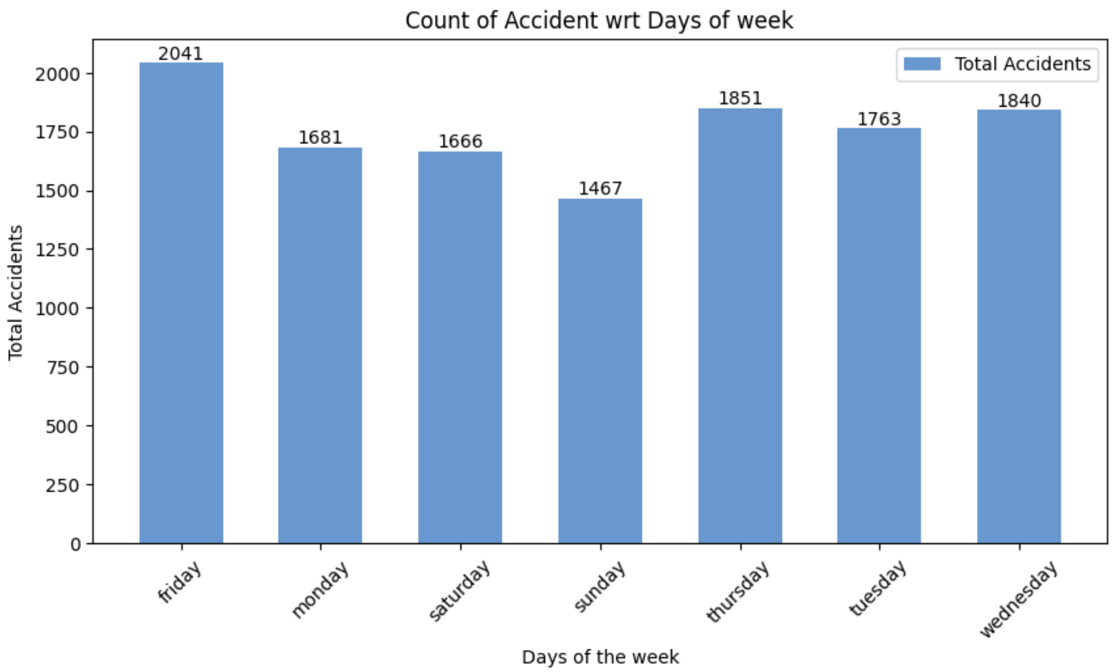
[137]: <matplotlib.legend.Legend at 0x7c08112946d0>

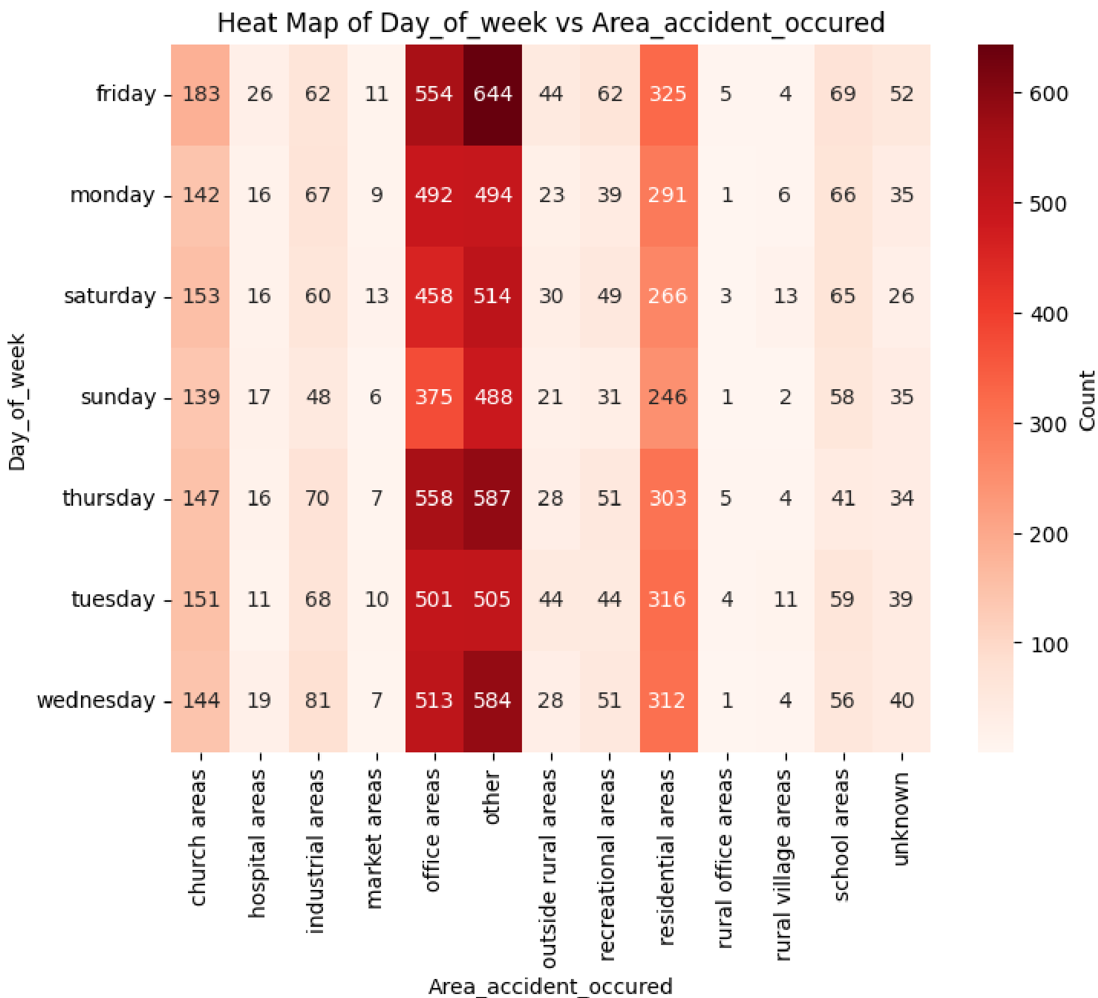


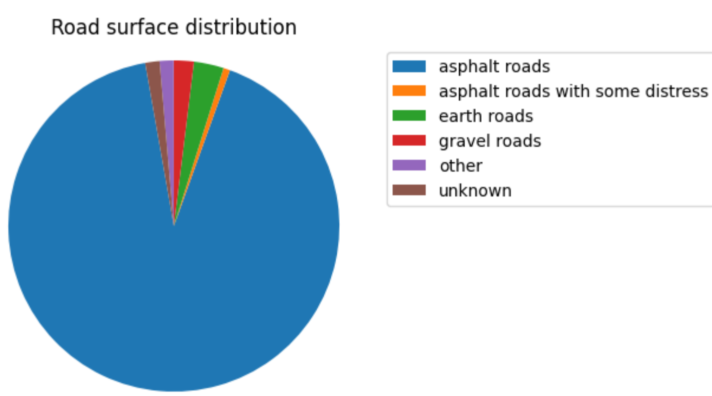
[138]:
```
plt.figure(figsize=(8, 5))
df3=cleaned_dataset.
  ↪groupby("Day_of_week",as_index=False)['Number_of_casualties'].count()
plt.figure(figsize=(10, 5))
bar_width = 0.6
hdi_bar =plt.bar(df3["Day_of_week"], df3["Number_of_casualties"],␣
  ↪width=bar_width, label='Total Accidents', color='#6997CF', align='center')
for bar in hdi_bar:
    death_cnt = bar.get_height()
    plt.text(bar.get_x() + bar.get_width() / 2, death_cnt, str(death_cnt),
             ha='center', va='bottom')
plt.legend(loc='upper right')
plt.xlabel('Days of the week')
plt.xticks(rotation=45)
plt.ylabel('Total Accidents')
plt.title('Count of Accident wrt Days of week')
plt.legend()
plt.show()

# Heat Map of Day_of_week vs Area_accident_occured
```

```
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Day_of_week"],␣
↪columns=df["Area_accident_occured"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Reds',␣
↪cbar_kws={'label': 'Count'})
plt.title('Heat Map of Day_of_week vs Area_accident_occured')
plt.xlabel('Area_accident_occured')
plt.ylabel('Day_of_week')
plt.show()

df=cleaned_dataset.
↪groupby("Road_surface_type",as_index=False)["Number_of_casualties"].count()
df.head(10)
# Pie Chart
plt.figure(figsize=(12, 4))
plt.pie(df['Number_of_casualties'], startangle=100)
plt.title('Road surface distribution')
plt.legend(df['Road_surface_type'], loc='upper right')
plt.axis('equal')
plt.show()
```

```
<Figure size 800x500 with 0 Axes>
```







Hypothesis 2: Understanding impact of light conditions on driving conditions through type of accidents, reason for accidents, weather condition

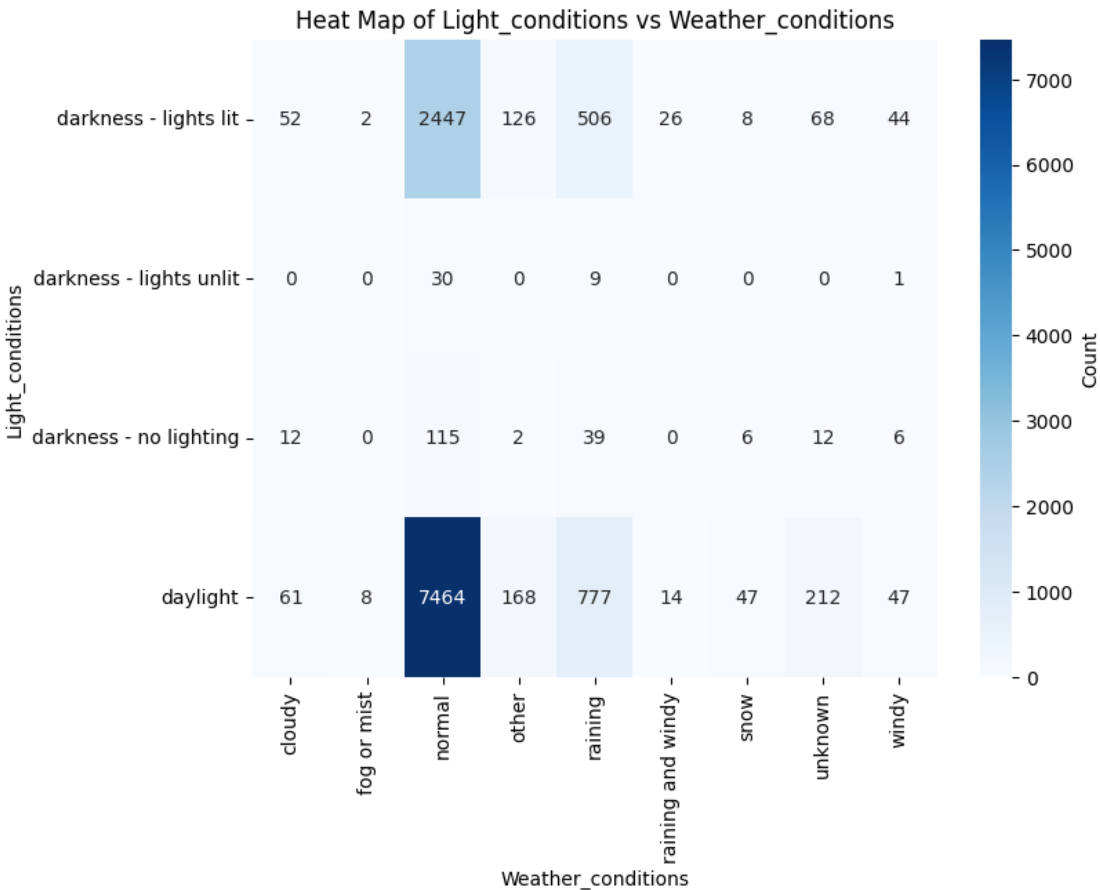
Heat Map of Light_conditions vs Weather_conditions

- Apart from Normal weather, Change in conditions like Raining, Cloudy, Snowy, Windy have impact on accidents. Amplifying more with Lighting conditions

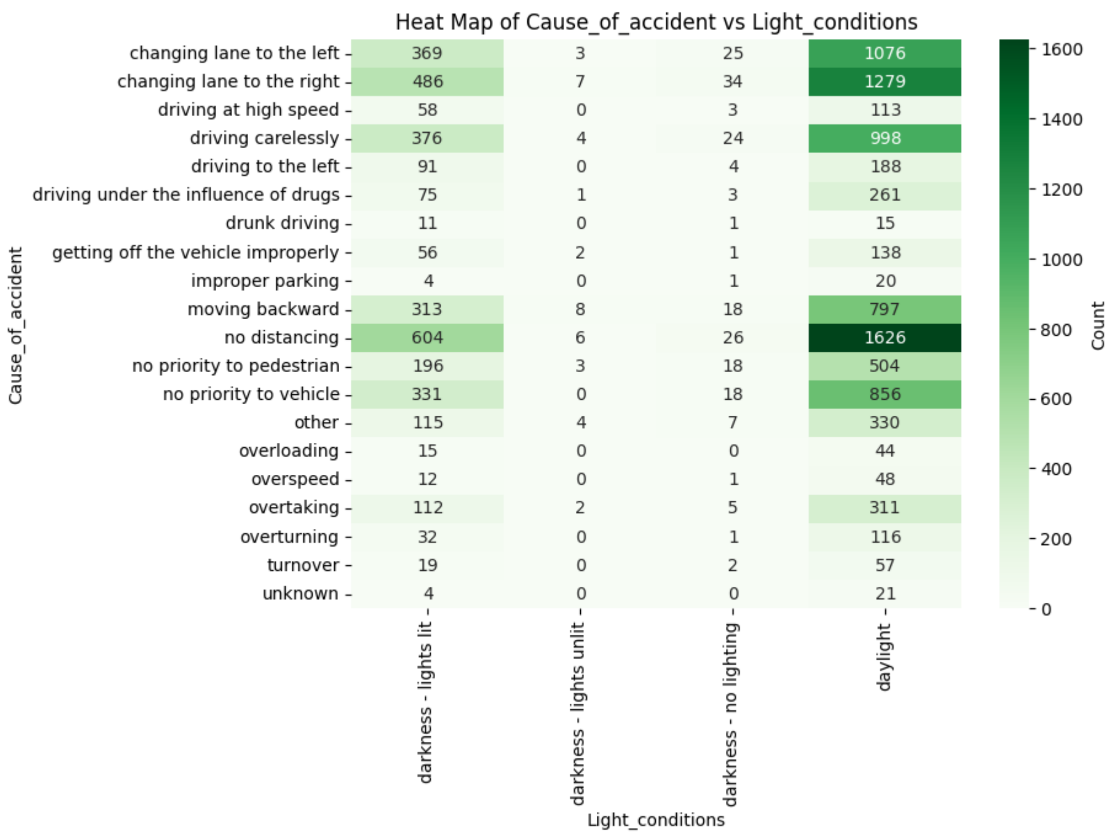
Heat Map of Cause_of_accident vs Light_conditions

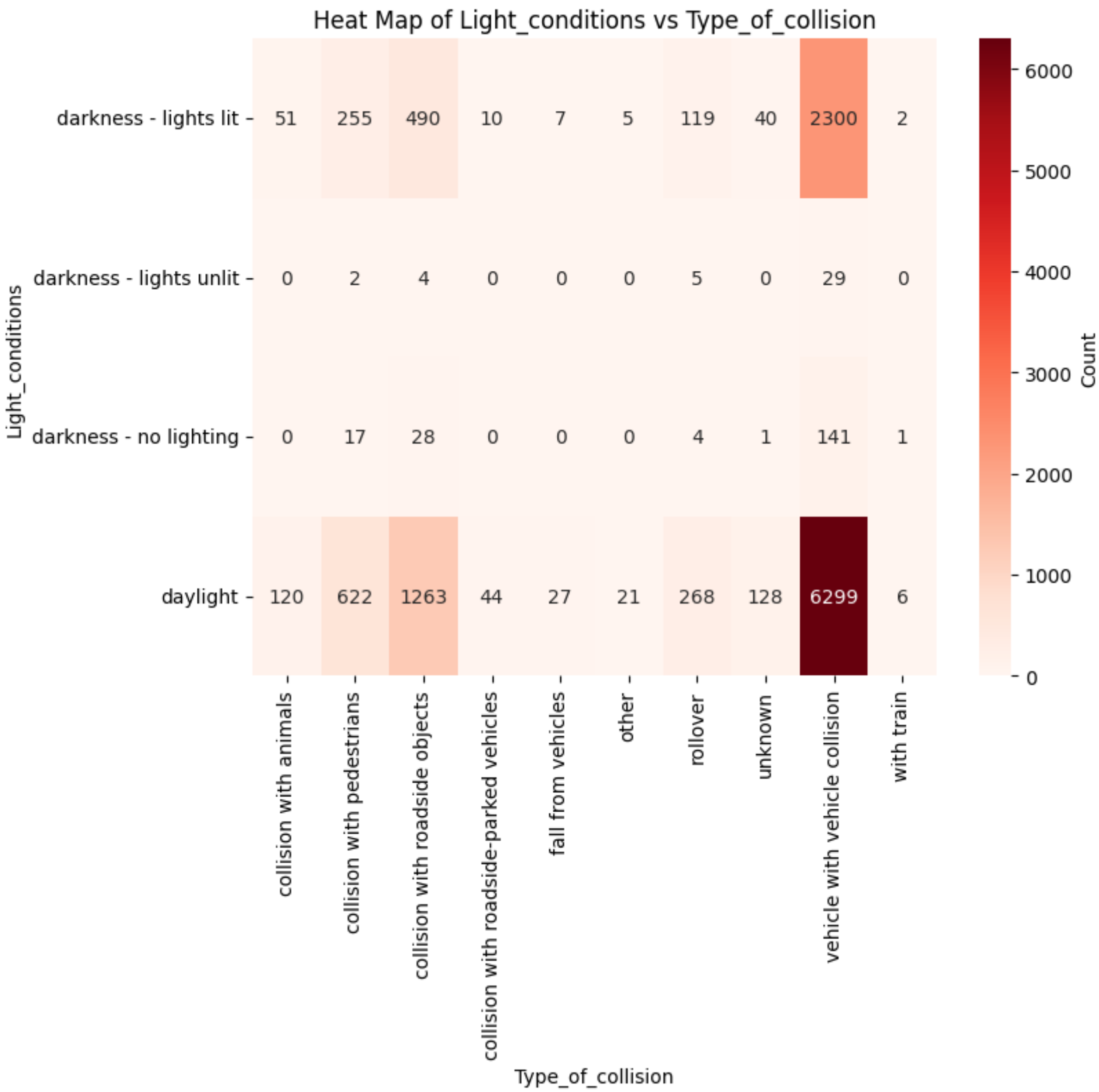
- Movement of vehicle is difficult to track in vehicle light and areas with no lighting. Count signifies the importance of properly-lit streets.

[139]:
```
#light conditions impact
# Heat Map of Light_conditions vs Weather_conditions
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Light_conditions"],␣
↪columns=df["Weather_conditions"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Blues',␣
↪cbar_kws={'label': 'Count'})
plt.title('Heat Map of Light_conditions vs Weather_conditions')
plt.xlabel('Weather_conditions')
plt.ylabel('Light_conditions')
plt.show()

# Heat Map of Cause_of_accident vs Light_conditions
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Cause_of_accident"],␣
↪columns=df["Light_conditions"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Greens',␣
↪cbar_kws={'label': 'Count'})
plt.title('Heat Map of Cause_of_accident vs Light_conditions')
plt.xlabel('Light_conditions')
plt.ylabel('Cause_of_accident')
plt.show()

# Heat Map of Light_conditions vs Type_of_collision
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Light_conditions"],␣
↪columns=df["Type_of_collision"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Reds',␣
↪cbar_kws={'label': 'Count'})
plt.title('Heat Map of Light_conditions vs Type_of_collision')
plt.xlabel('Type_of_collision')
plt.ylabel('Light_conditions')
plt.show()
```



Heat Map of Light_conditions vs Weather_conditions
Light_conditions
darkness - lights lit
darkness - lights unlit
darkness - no lighting
daylight
52
2
2447
126
506
26
8
68
44
0
0
30
0
9
0
0
0
1
12
0
115
2
39
0
6
12
6
61
8
7464
168
777
14
47
212
47
cloudy
fog or mist
normal
other
raining
raining and windy
snow
unknown
windy
Weather_conditions
Count
7000
6000
5000
4000
3000
2000
1000
0


Heat Map of Cause_of_accident vs Light_conditions
Cause_of_accident
changing lane to the left
changing lane to the right
driving at high speed
driving carelessly
driving to the left
driving under the influence of drugs
drunk driving
getting off the vehicle improperly
improper parking
moving backward
no distancing
no priority to pedestrian
no priority to vehicle
other
overloading
overspeed
overtaking
overturning
turnover
unknown
Light_conditions
darkness - lights lit
darkness - lights unlit
darkness - no lighting
daylight
Count
369 3 25 1076
486 7 34 1279
58 0 3 113
376 4 24 998
91 0 4 188
75 1 3 261
11 0 1 15
56 2 1 138
4 0 1 20
313 8 18 797
604 6 26 1626
196 3 18 504
331 0 18 856
115 4 7 330
15 0 0 44
12 0 1 48
112 2 5 311
32 0 1 116
19 0 2 57
4 0 0 21
0
200
400
600
800
1000
1200
1400
1600



Hypothesis 2: Understanding impact of Weather conditions on driving conditions through type of accidents, reason for accidents, weather condition

Heat Map of Vehicle_movement vs Weather_conditions

- Bad Driving practices amplify with non normal environment conditions. we can see that from map.

Heat Map of Weather_conditions vs Type_of_collision

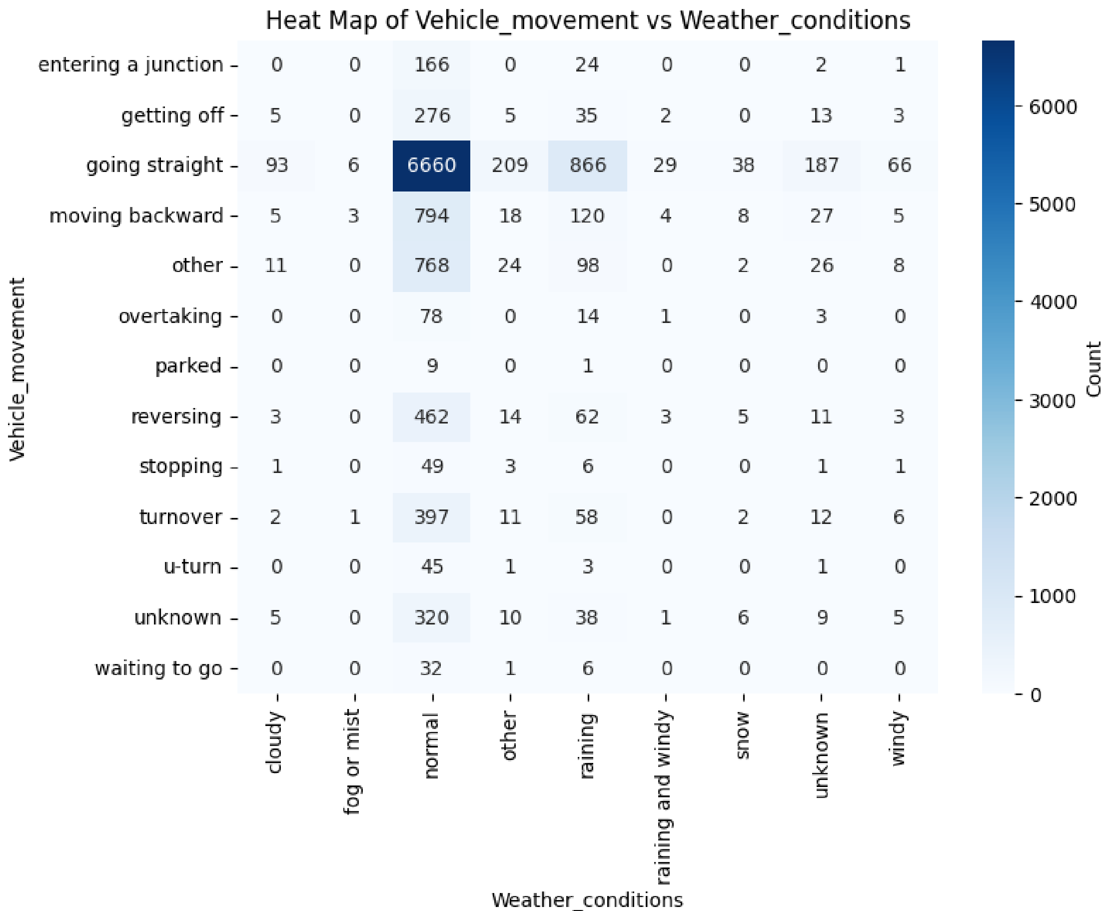
- Lack of proper lighting has resulted in collision with pedestrians and also road side objects.
- Even under car lights we see some accidents, suggesting improvement in lighting of certain areas

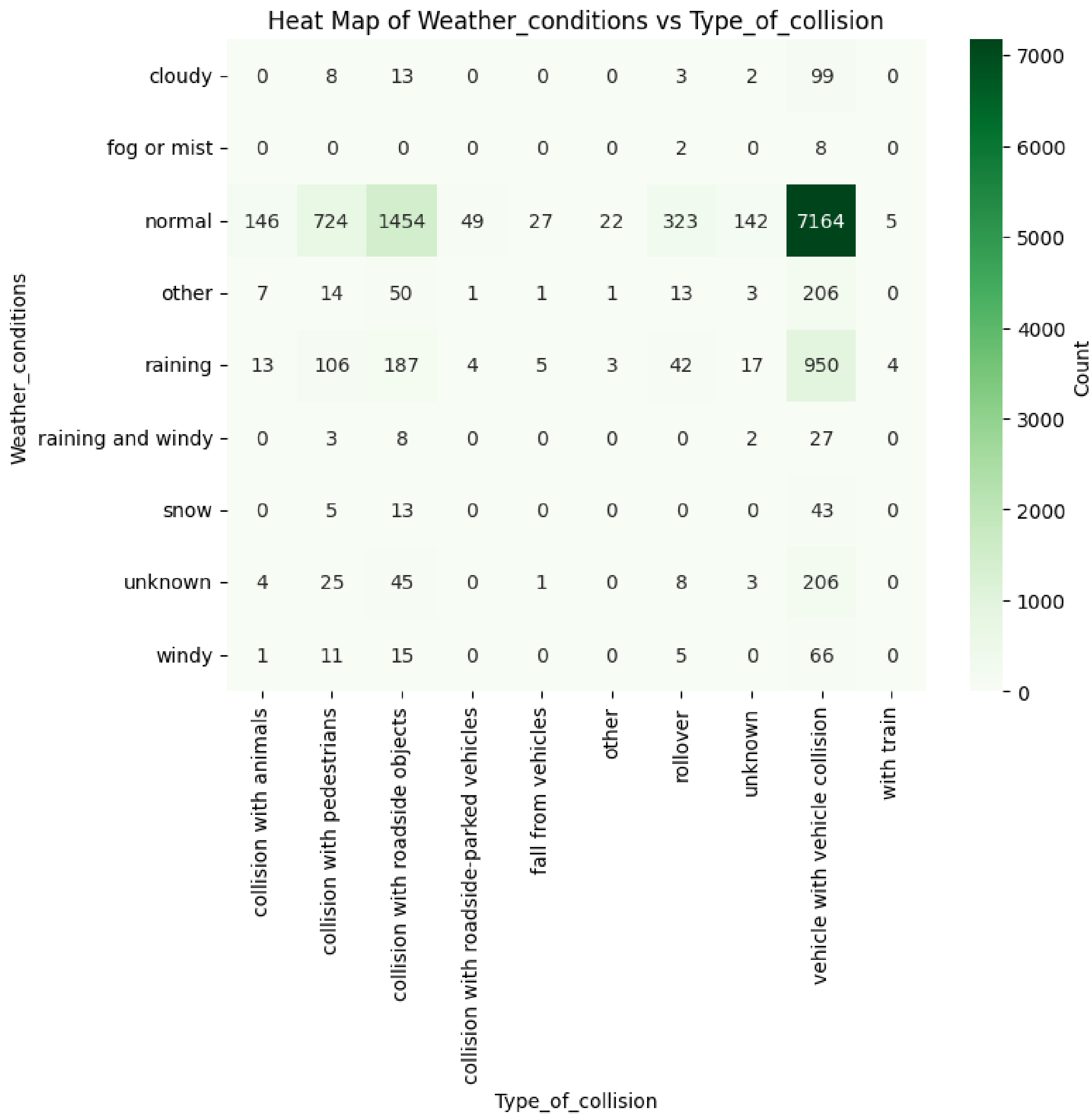
Heat Map of Cause_of_accident vs Weather_conditions

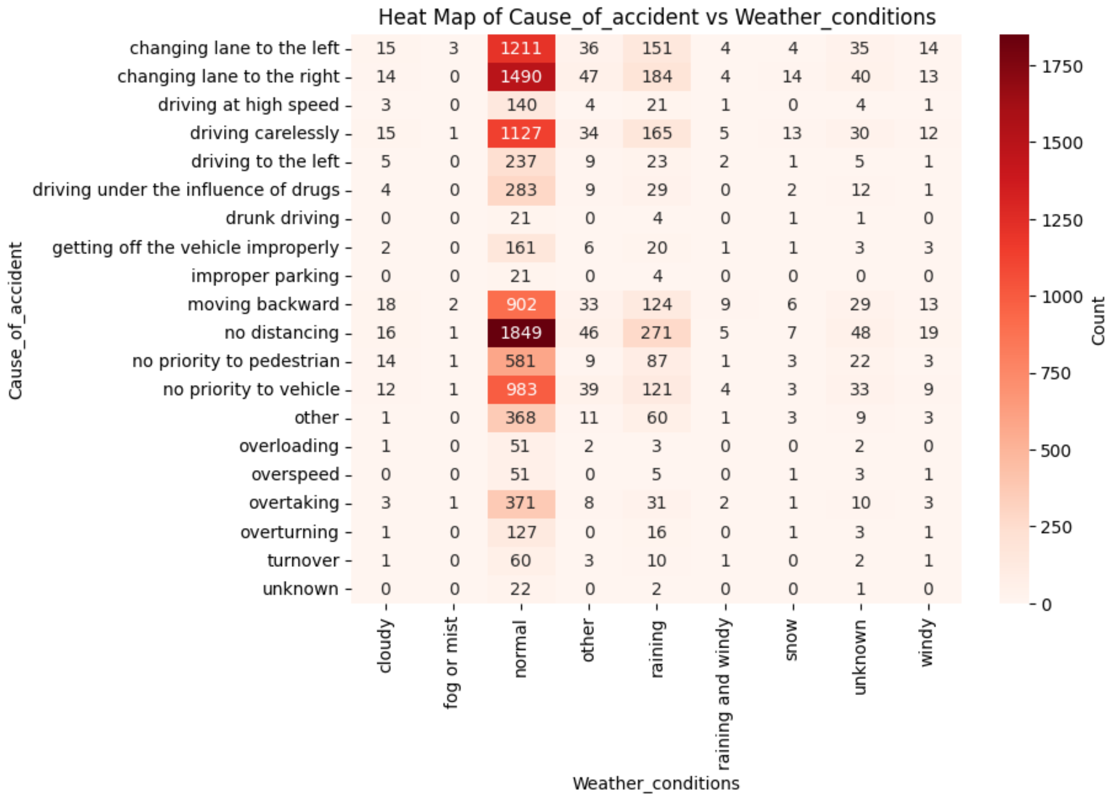
- Climatic conditions lead drivers to induce mistakes due to lack of clear vision, less road surface grip and lower control over vehicle

- We see rain and fog have induced driving errors from the distribution

[140]:
```
# Distribution of  details with Weather_conditions
#Heat Map of Vehicle_movement vs Weather_conditions
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Vehicle_movement"],␣
 ↪columns=df["Weather_conditions"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Blues',␣
 ↪cbar_kws={'label': 'Count'})
plt.title('Heat Map of Vehicle_movement vs Weather_conditions')
plt.xlabel('Weather_conditions')
plt.ylabel('Vehicle_movement')
plt.show()

#Heat Map of Weather_conditions vs Type_of_collision
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Weather_conditions"],␣
 ↪columns=df["Type_of_collision"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Greens',␣
 ↪cbar_kws={'label': 'Count'})
plt.title('Heat Map of Weather_conditions vs Type_of_collision')
plt.xlabel('Type_of_collision')
plt.ylabel('Weather_conditions')
plt.show()

# Heat Map of Cause_of_accident vs Weather_conditions
df = pd.DataFrame(cleaned_dataset)
distribution_table = df.pivot_table(index=df["Cause_of_accident"],␣
 ↪columns=df["Weather_conditions"], aggfunc='size', fill_value=0)
plt.figure(figsize=(8, 6))
sns.heatmap(distribution_table, annot=True, fmt='d', cmap='Reds',␣
 ↪cbar_kws={'label': 'Count'})
plt.title('Heat Map of Cause_of_accident vs Weather_conditions')
plt.xlabel('Weather_conditions')
plt.ylabel('Cause_of_accident')
plt.show()
```


Heat Map of Vehicle_movement vs Weather_conditions
Vehicle_movement
Weather_conditions
Count
entering a junction
getting off
going straight
moving backward
other
overtaking
parked
reversing
stopping
turnover
u-turn
unknown
waiting to go
cloudy
fog or mist
normal
other
raining
raining and windy
snow
unknown
windy
0 0 166 0 24 0 0 2 1
5 0 276 5 35 2 0 13 3
93 6 6660 209 866 29 38 187 66
5 3 794 18 120 4 8 27 5
11 0 768 24 98 0 2 26 8
0 0 78 0 14 1 0 3 0
0 0 9 0 1 0 0 0 0
3 0 462 14 62 3 5 11 3
1 0 49 3 6 0 0 1 1
2 1 397 11 58 0 2 12 6
0 0 45 1 3 0 0 1 0
5 0 320 10 38 1 6 9 5
0 0 32 1 6 0 0 0 0
0
1000
2000
3000
4000
5000
6000


Heat Map of Weather_conditions vs Type_of_collision
cloudy
fog or mist
normal
other
raining
raining and windy
snow
unknown
windy
Weather_conditions
collision with animals
collision with pedestrians
collision with roadside objects
collision with roadside-parked vehicles
fall from vehicles
other
rollover
unknown
vehicle with vehicle collision
with train
Type_of_collision
Count
0 8 13 0 0 0 3 2 99 0
0 0 0 0 0 0 2 0 8 0
146 724 1454 49 27 22 323 142 7164 5
7 14 50 1 1 1 13 3 206 0
13 106 187 4 5 3 42 17 950 4
0 3 8 0 0 0 0 2 27 0
0 5 13 0 0 0 0 0 43 0
4 25 45 0 1 0 8 3 206 0
1 11 15 0 0 0 5 0 66 0
7000
6000
5000
4000
3000
2000
1000
0



	cloudy	fog or mist	normal	other	raining	raining and windy	snow	unknown	windy
changing lane to the left	15	3	1211	36	151	4	4	35	14
changing lane to the right	14	0	1490	47	184	4	14	40	13
driving at high speed	3	0	140	4	21	1	0	4	1
driving carelessly	15	1	1127	34	165	5	13	30	12
driving to the left	5	0	237	9	23	2	1	5	1
driving under the influence of drugs	4	0	283	9	29	0	2	12	1
drunk driving	0	0	21	0	4	0	1	1	0
getting off the vehicle improperly	2	0	161	6	20	1	1	3	3
improper parking	0	0	21	0	4	0	0	0	0
moving backward	18	2	902	33	124	9	6	29	13
no distancing	16	1	1849	46	271	5	7	48	19
no priority to pedestrian	14	1	581	9	87	1	3	22	3
no priority to vehicle	12	1	983	39	121	4	3	33	9
other	1	0	368	11	60	1	3	9	3
overloading	1	0	51	2	3	0	0	2	0
overspeed	0	0	51	0	5	0	1	3	1
overtaking	3	1	371	8	31	2	1	10	3
overturning	1	0	127	0	16	0	1	3	1
turnover	1	0	60	3	10	1	0	2	1
unknown	0	0	22	0	2	0	0	1	0