In [73]:
```python
#Load essential libraries for processing and visualising data

import numpy as np
import pandas as pd             #Processing data
import matplotlib.pyplot as plt   #Visualisation
import seaborn as sns           #Visualisation
```

In [74]:
```python
df = pd.read_excel('NSW_Road_Crash_Data_2017-2021_CRASH.xlsx')
df.head()
```

Out[74]:

	CrashID	Degree_of_crash	Degree_of_crash_detailed	Reporting_year	Year_of_crash	Month_of_crash	Day_of_week_of_crash	Two_hour_intervals	Street_of_cra
0	1122708	Fatal	Fatal	2017.0	2017.0	January	Monday	18:00 - 19:59	HOLLOWA
1	1122709	Fatal	Fatal	2017.0	2017.0	January	Monday	12:00 - 13:59	PUT
2	1122710	Fatal	Fatal	2017.0	2017.0	January	Tuesday	14:00 - 15:59	IRRIGATIC
3	1123942	Fatal	Fatal	2017.0	2017.0	January	Thursday	10:00 - 11:59	VARD
4	1123948	Fatal	Fatal	2017.0	2017.0	January	Saturday	12:00 - 13:59	PRINC

5 rows × 49 columns

In [4]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101718 entries, 0 to 101717
Data columns (total 49 columns):
 #   Column                        Non-Null Count   Dtype  
---  ------                        --------------   -----  
 0   CrashID                       101718 non-null  int64  
 1   Degree_of_crash               101718 non-null  object 
 2   Degree_of_crash_detailed      101714 non-null  object 
 3   Reporting_year                101714 non-null  float64
 4   Year_of_crash                 101714 non-null  float64
 5   Month_of_crash                101714 non-null  object 
 6   Day_of_week_of_crash          101713 non-null  object 
 7   Two_hour_intervals            101713 non-null  object 
 8   Street_of_crash               101714 non-null  object 
 9   Street_type                   101714 non-null  object 
 10  Distance                      101715 non-null  float64
 11  Direction                     101714 non-null  object 
 12  Identifying_feature           101714 non-null  object 
 13  Identifying_feature_type      101718 non-null  object 
 14  Town                          101717 non-null  object 
 15  Route_no                      64996 non-null   float64
 16  School_zone_location          101717 non-null  object 
 17  School_zone_active            101717 non-null  object 
 18  Type_of_location              101716 non-null  object 
 19  Latitude                      101715 non-null  float64
 20  Longitude                     101715 non-null  float64
 21  LGA                           101715 non-null  object 
 22  Urbanisation                  101716 non-null  object 
 23  Alignment                     101716 non-null  object 
 24  Primary_permanent_feature     24733 non-null   object 
 25  Primary_temporary_feature     1487 non-null    object 
 26  Primary_hazardous_feature     2045 non-null    object 
 27  Street_lighting               101717 non-null  object 
 28  Road_surface                  101717 non-null  object 
 29  Surface_condition             101717 non-null  object 
 30  Weather                       101717 non-null  object 
 31  Natural_lighting              101718 non-null  object 
 32  Signals_operation             101718 non-null  object 
 33  Other_traffic_control         101718 non-null  object 
 34  Speed_limit                   101718 non-null  object 
 35  Road_classification           101718 non-null  object 
 36  RUM_code                      101718 non-null  int64  
 37  RUM_description               101718 non-null  object 
 38  DCA_code                      101718 non-null  int64  
 39  DCA_description               101718 non-null  object 
 40  DCA_supplement                12695 non-null   object 
 41  First_impact_type             101718 non-null  object 
 42  Key_TU_type                   101718 non-null  object 
 43  Other_TU_type                 72060 non-null   object 
 44  No_of_traffic_units_involved  101718 non-null  int64  
 45  No_killed                     101718 non-null  int64  
 46  No_seriously_injured          101718 non-null  int64  
 47  No_moderately_injured         101718 non-null  int64  
 48  No_minor_other_injured        101718 non-null  int64  
dtypes: float64(6), int64(8), object(35)
memory usage: 38.0+ MB
```

Data Cleaning

In [84]:
```python
#Categorical columns
cat_cols = df.select_dtypes(include=['object', 'category']).columns
```

```
cat_cols
```

Out[84]:
```
Index(['Degree_of_crash', 'Degree_of_crash_detailed', 'Month_of_crash',
       'Day_of_week_of_crash', 'Two_hour_intervals', 'Street_of_crash',
       'Street_type', 'Direction', 'Identifying_feature',
       'Identifying_feature_type', 'Town', 'School_zone_location',
       'School_zone_active', 'Type_of_location', 'LGA', 'Urbanisation',
       'Alignment', 'Primary_permanent_feature', 'Primary_temporary_feature',
       'Primary_hazardous_feature', 'Street_lighting', 'Road_surface',
       'Surface_condition', 'Weather', 'Natural_lighting', 'Signals_operation',
       'Other_traffic_control', 'Speed_limit', 'Road_classification',
       'RUM_description', 'DCA_description', 'DCA_supplement',
       'First_impact_type', 'Key_TU_type', 'Other_TU_type'],
      dtype='object')
```

In [85]:
```
#Replace null values in categorical columns with the mode
df[cat_cols] = df[cat_cols].apply(lambda x: x.fillna(x.mode()[0]))

print(df.isnull().sum())
```

```
CrashID                          0
Degree_of_crash                  0
Degree_of_crash_detailed         0
Reporting_year                   4
Year_of_crash                    4
Month_of_crash                   0
Day_of_week_of_crash             0
Two_hour_intervals               0
Street_of_crash                  0
Street_type                      0
Distance                         3
Direction                        0
Identifying_feature              0
Identifying_feature_type         0
Town                             0
Route_no                     36722
School_zone_location             0
School_zone_active               0
Type_of_location                 0
Latitude                         3
Longitude                        3
LGA                              0
Urbanisation                     0
Alignment                        0
Primary_permanent_feature        0
Primary_temporary_feature        0
Primary_hazardous_feature        0
Street_lighting                  0
Road_surface                     0
Surface_condition                0
Weather                          0
Natural_lighting                 0
Signals_operation                0
Other_traffic_control            0
Speed_limit                      0
Road_classification              0
RUM_code                         0
RUM_description                  0
DCA_code                         0
DCA_description                  0
DCA_supplement                   0
First_impact_type                0
Key_TU_type                      0
Other_TU_type                    0
No_of_traffic_units_involved     0
No_killed                        0
No_seriously_injured             0
No_moderately_injured            0
No_minor_other_injured           0
Crash_severe                     0
dtype: int64
```

In [86]:
```
#Numerical_columns
numerical_cols = df.select_dtypes(include=['float64', 'int']).columns
numerical_cols
```

Out[86]:
```
Index(['CrashID', 'Reporting_year', 'Year_of_crash', 'Distance', 'Route_no',
       'Latitude', 'Longitude', 'RUM_code', 'DCA_code',
       'No_of_traffic_units_involved', 'No_killed', 'No_seriously_injured',
       'No_moderately_injured', 'No_minor_other_injured', 'Crash_severe'],
      dtype='object')
```

In [87]:
```
#Replace null values in numerical columns with the mean
df[numerical_cols] = df[numerical_cols].apply(lambda x: x.fillna(x.mean()))

print(df.isnull().sum())
```

```
CrashID                         0
Degree_of_crash                 0
Degree_of_crash_detailed        0
Reporting_year                  0
Year_of_crash                   0
Month_of_crash                  0
Day_of_week_of_crash            0
Two_hour_intervals              0
Street_of_crash                 0
Street_type                     0
Distance                        0
Direction                       0
Identifying_feature             0
Identifying_feature_type        0
Town                            0
Route_no                        0
School_zone_location            0
School_zone_active              0
Type_of_location                0
Latitude                        0
Longitude                       0
LGA                             0
Urbanisation                    0
Alignment                       0
Primary_permanent_feature       0
Primary_temporary_feature       0
Primary_hazardous_feature       0
Street_lighting                 0
Road_surface                    0
Surface_condition               0
Weather                         0
Natural_lighting                0
Signals_operation               0
Other_traffic_control           0
Speed_limit                     0
Road_classification             0
RUM_code                        0
RUM_description                 0
DCA_code                        0
DCA_description                 0
DCA_supplement                  0
First_impact_type               0
Key_TU_type                     0
Other_TU_type                   0
No_of_traffic_units_involved    0
No_killed                       0
No_seriously_injured            0
No_moderately_injured           0
No_minor_other_injured          0
Crash_severe                    0
dtype: int64
```

In [88]:

```
df.duplicated().sum() #Check duplicates
```

Out[88]:

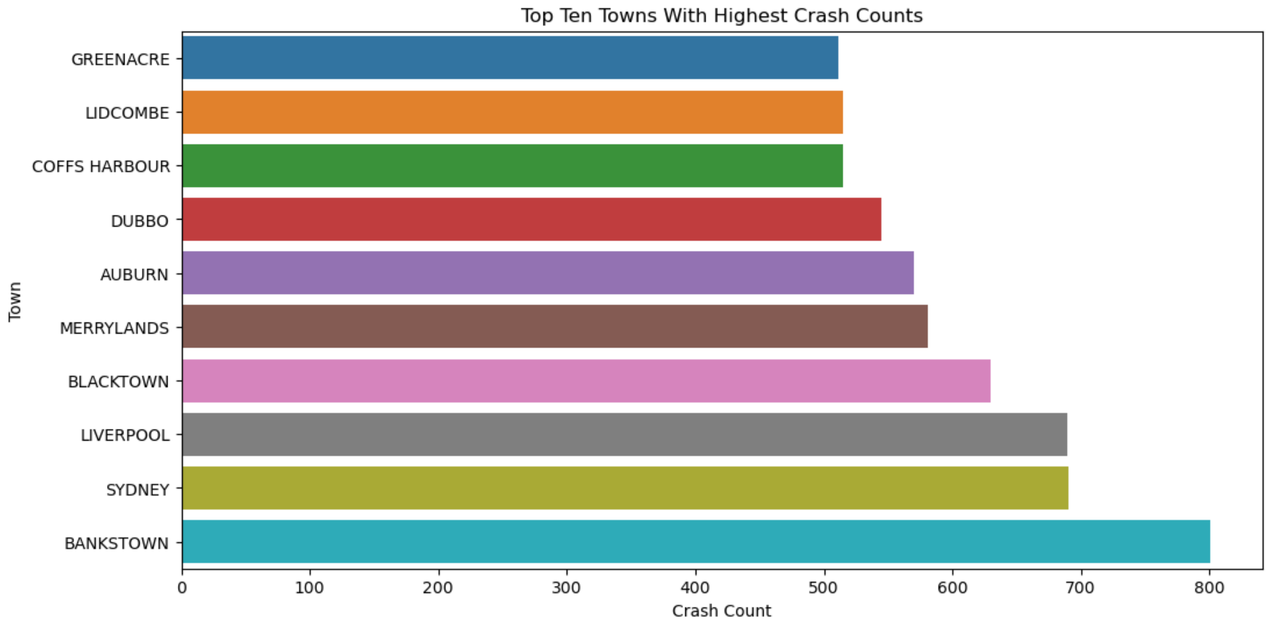
```
0
```

Visualisation

Bar chart - top five towns/suburbs with highest no. of crashes

In [15]:

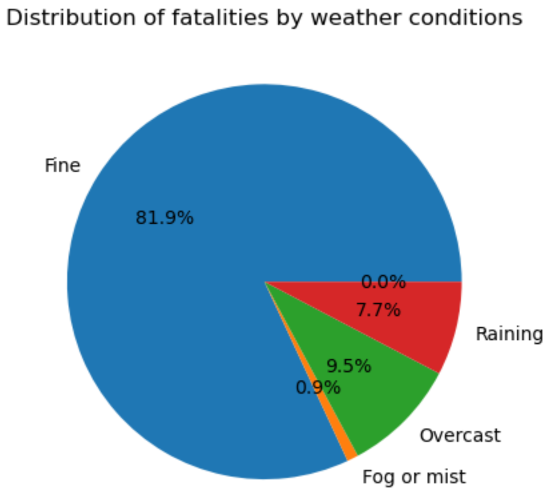
```
top_10_towns = df.groupby('Town')['CrashID'].count().sort_values(ascending=False).head(10) #Select 10 towns with highest accident/cra
top_10_towns_reorder = top_10_towns.sort_values(ascending=True)  #order them in ascending order

plt.figure(figsize=(12, 6)) #fix the visual size

sns.barplot(x=top_10_towns_reorder.values, y=top_10_towns_reorder.index, orient='h')  #seaborn horizontal bar plot

plt.xlabel('Crash Count')
plt.ylabel('Town')
plt.title('Top Ten Towns With Highest Crash Counts')

plt.show()
```



Pie chart - display the proportion of fatalities occurring in different weathers

In [22]:
```
plt.style.use('classic')
plt.style.use('default')

# Filter out data from the year 2016
df1 = df[df['Reporting_year'] != 2016]

# Filter out Unknown weather
df1 = df1[df1['Weather'] != 'Unknown']
df1 = df1[df1['Weather'] != 'Other']


# Remove duplicates based on specific columns
d1 = df1.drop_duplicates(subset=['CrashID'])

plt.figure(figsize=(12, 6))
df1.groupby('Weather').sum().plot(kind='pie',
                                                title ='Distribution of fatalities by weather conditions',
                                                y='No_killed',  legend = False,
                                                ylabel = '', autopct='%1.1f%%')

plt.figure(figsize=(12, 6)) #fix the visual size

plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```

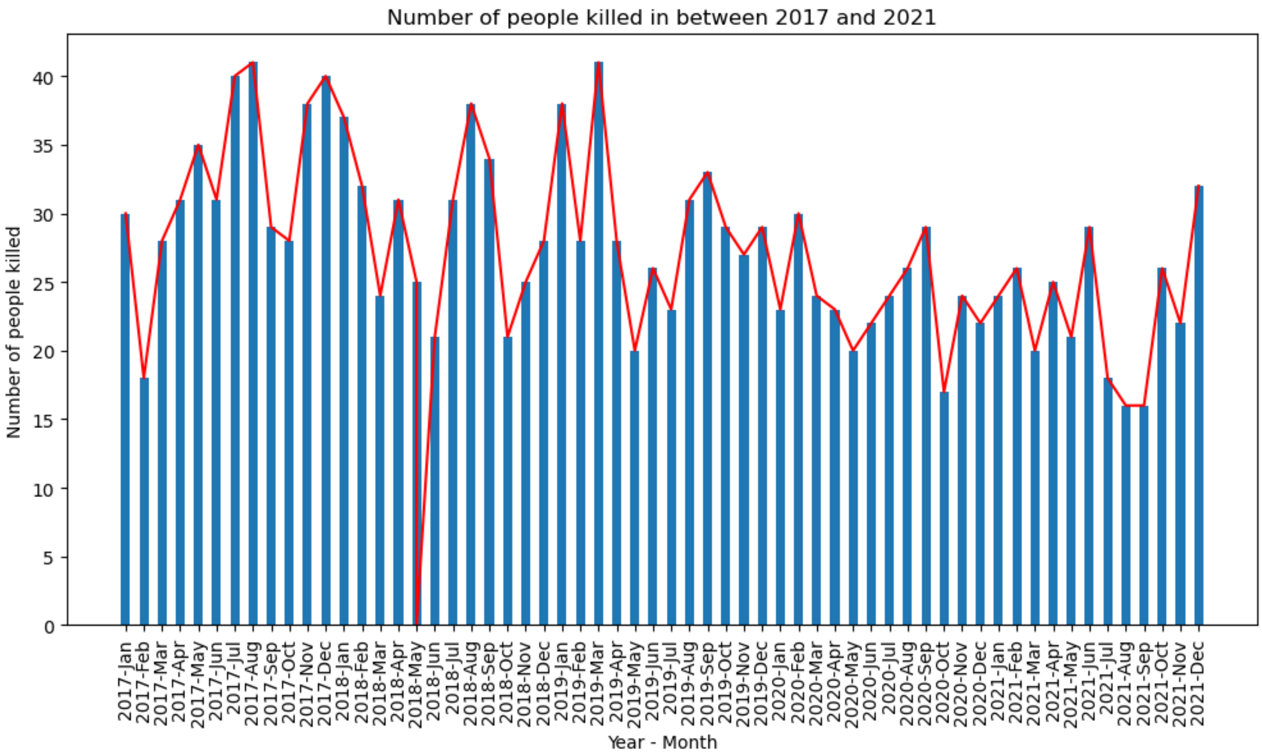



```
<Figure size 1200x600 with 0 Axes>
```

Line plot & bar chart - distribution of the number of people killed on the road vary by month & year

In [23]:
```
#create new dataframe for grouped data
grouped_data = df.groupby(['Year_of_crash', 'Month_of_crash'])['No_killed'].sum()
grouped_data = grouped_data.reset_index()

#Clean out rows where the year is 2016 or 0
grouped_data['Year_of_crash'] = grouped_data['Year_of_crash'].fillna(0).astype(int)
grouped_data = grouped_data[(grouped_data['Year_of_crash'] != 2016) & (grouped_data['Year_of_crash'] != 0)]
```

```
#creating format for x axis
grouped_data['Year-Month'] = grouped_data['Year_of_crash'].astype(str).str[:4] + '-' + grouped_data['Month_of_crash'].str[:3]

#map out the months using a dictionary to order it later
month_dict = {'January': 1, 'February': 2, 'March': 3, 'April': 4, 'May': 5, 'June': 6, 'July': 7, 'August': 8, 'September': 9, 'Octo
grouped_data['Month_Number'] = grouped_data['Month_of_crash'].map(month_dict)

#Sort the data by the 'Year and Month_Number' column
grouped_data = grouped_data.sort_values(['Year_of_crash', 'Month_Number'])

#creating line
plt.figure(figsize=(12, 6))
plt.plot(grouped_data['Year-Month'], grouped_data['No_killed'], color = 'r')

#creating bar
plt.bar(grouped_data['Year-Month'], grouped_data['No_killed'], width=0.5)
plt.xticks(rotation = 90)
plt.xlabel("Year - Month")
plt.ylabel("Number of people killed")
plt.title("Number of people killed in between 2017 and 2021")
plt.show()
```



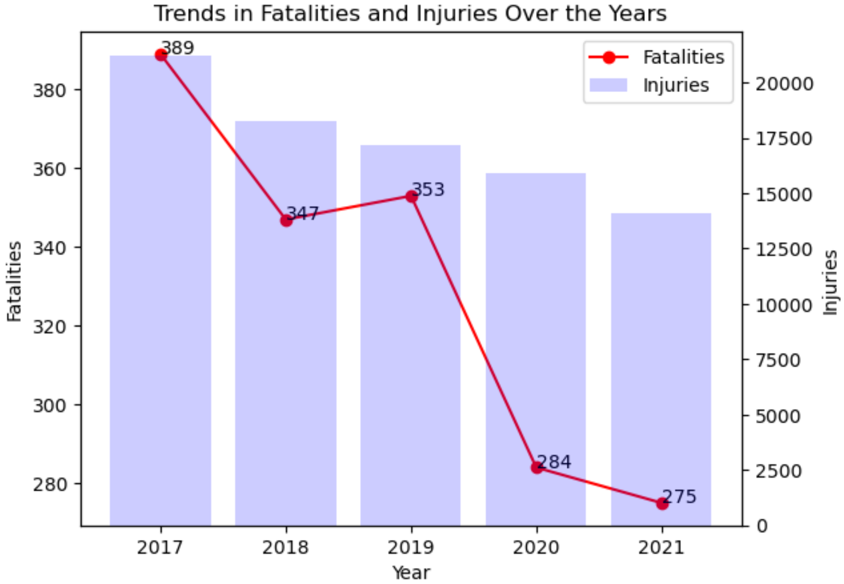
Line plot & bar chart - show the trend in the number of fatalities and injuries over years

In [25]:

```
df['Year_of_crash'] = df['Year_of_crash'].astype(int)
#change dtype of year_of_crash column
df.dtypes #check if the dtype changed

#Total fatalities over year
year_total_fatal = df.groupby('Year_of_crash')['No_killed'].sum().reset_index()
year_total_fatal = year_total_fatal[(year_total_fatal['Year_of_crash'] != 0) & (year_total_fatal['Year_of_crash'] != 2016)]   #remove
year_total_fatal

#Total injuries over year
year_total_injury = df.groupby('Year_of_crash')[['No_seriously_injured', 'No_moderately_injured', 'No_minor_other_injured']].sum().re
year_total_injury['Total_Injuries'] = year_total_injury[['No_seriously_injured', 'No_moderately_injured', 'No_minor_other_injured']].
year_total_injury = year_total_injury[(year_total_injury['Year_of_crash'] != 0) & (year_total_injury['Year_of_crash'] != 2016)]   #re
year_total_injury

import matplotlib.pyplot as plt  #import visualisation library

year = year_total_fatal['Year_of_crash']  #select values from year_of_crash column in year_total_fatal dataframe for plotting

total_injuries = year_total_injury['Total_Injuries']   #select values from total_injuries column in year_total_fatal dataframe for pl

total_fatalities = year_total_fatal['No_killed']   #select values from no_killed column in year_total_fatal dataframe for plotting

plt.figure(figsize=(12,6))  #define plot size
fig, ax1 = plt.subplots()


ax1.set_xlabel('Year')

#Create line plot for total fatalities --- total fatalities on the left y-axis
ax1.set_ylabel('Fatalities', color='black')
ax1.plot(year, total_fatalities, marker='o', color = 'red', label='Fatalities')

#Create barplot for total injuries -- total injuries on the right y-axis
```

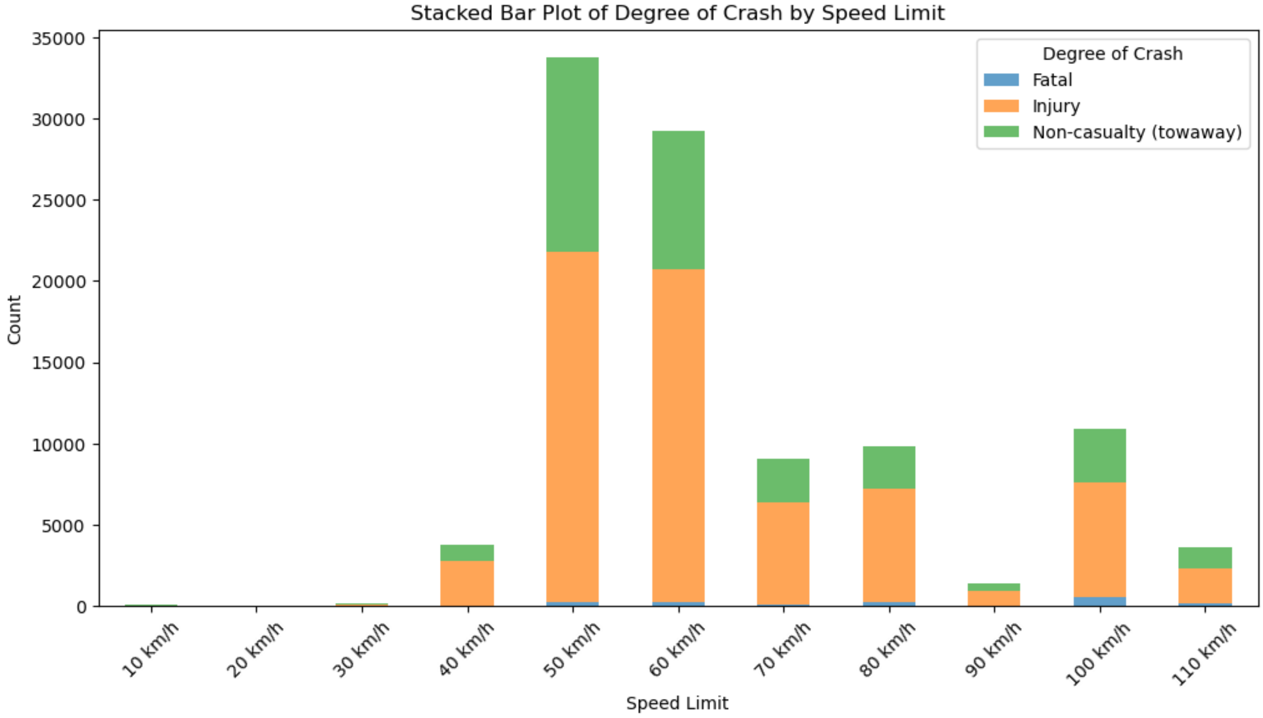
```
ax2 = ax1.twinx()
ax2.set_ylabel('Injuries', color='black')
ax2.bar(year, total_injuries, color = 'blue', alpha = 0.2, label='Injuries')

# Display the fatality numbers
for i,j in zip(year, total_fatalities):
    ax1.annotate(str(j),xy=(i,j))

plt.title("Trends in Fatalities and Injuries Over the Years")


# Add a single legend for both bar and line plot
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
lines = lines + lines2
labels = labels + labels2

# Display the combined legend in the upper right
ax2.legend(lines, labels, loc = 'upper right')

#Show the plot
plt.show()
```

```
<Figure size 1200x600 with 0 Axes>
```



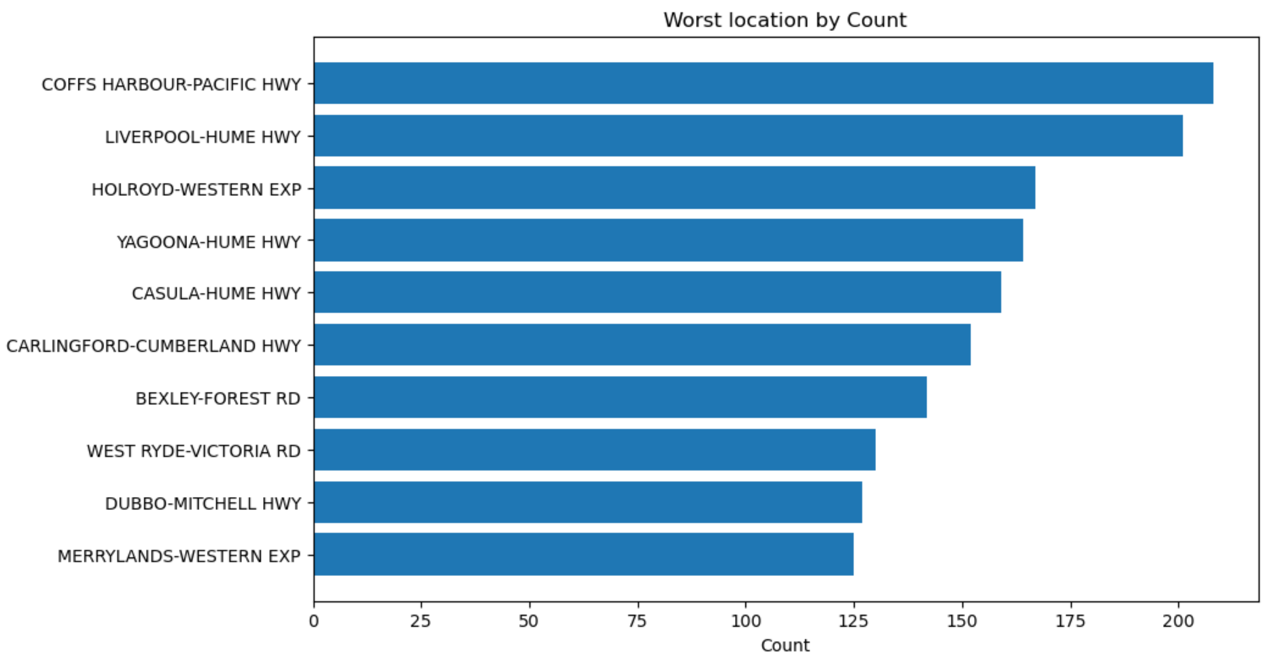
Stacked bar chart - relationship between the speed limit and accident statistics

In [26]:
```
# Define the desired order of speed limits
speed_limit_order = [
    '10 km/h', '20 km/h', '30 km/h',
    '40 km/h', '50 km/h', '60 km/h',
    '70 km/h', '80 km/h', '90 km/h',
    '100 km/h', '110 km/h'
]

# Convert 'Speed_Limit' to categorical with the desired order
df['Speed_limit'] = pd.Categorical(
    df['Speed_limit'], categories=speed_limit_order, ordered=True)


# Group the data by 'Speed_Limit' and 'Degree_of_crash' and count occurrences
grouped_data = df.groupby(['Speed_limit', 'Degree_of_crash']).size().unstack()

# Create a stacked bar chart
ax = grouped_data.plot(kind='bar', stacked=True, figsize=(12, 6), alpha=0.7)  # Adjust alpha for visibility

# Set labels and title
plt.xlabel("Speed Limit")
plt.ylabel("Count")
plt.title("Stacked Bar Plot of Degree of Crash by Speed Limit")

# Show the legend
plt.legend(title="Degree of Crash")

# Rotate x-axis labels for better readability
ax.set_xticklabels(ax.get_xticklabels(), rotation=45)

# Show the plot
plt.show()
```



Bar chart - show most notorious locations in the past 5 years

In [28]:
```python
df6 = pd.DataFrame()
df6['Town'] = df['Town']
df6['Street_of_crash'] = df['Street_of_crash']
df6['Street_type'] = df['Street_type']
df6['Year_of_crash'] = df['Year_of_crash']

#Concat columns
df6['Concatenated'] = df6['Town'] + '-' + df6['Street_of_crash'] + ' ' + df6['Street_type']

#Clean out rows where the year is 2016 or 0
df6['Year_of_crash'] = df6['Year_of_crash'].astype(int)
df6 = df6[(df6['Year_of_crash'] != 2016) & (df6['Year_of_crash'] != 0)]

#Group by 'Concatenated' and count the number of occurrences
grouped_df6 = df6.groupby('Concatenated').size().reset_index(name='count')

#Select the top 10 locations
grouped_df6 = grouped_df6.sort_values('count', ascending=True).tail(10)

#Create a horizontal bar chart
plt.figure(figsize=(10, 6))
plt.barh(grouped_df6['Concatenated'], grouped_df6['count'])
plt.xlabel('Count')
plt.title('Worst location by Count')
plt.show()
```



Density heatmap - show crashes in geographical locations

In [30]:
```
pip install dash  #Install dash library
```

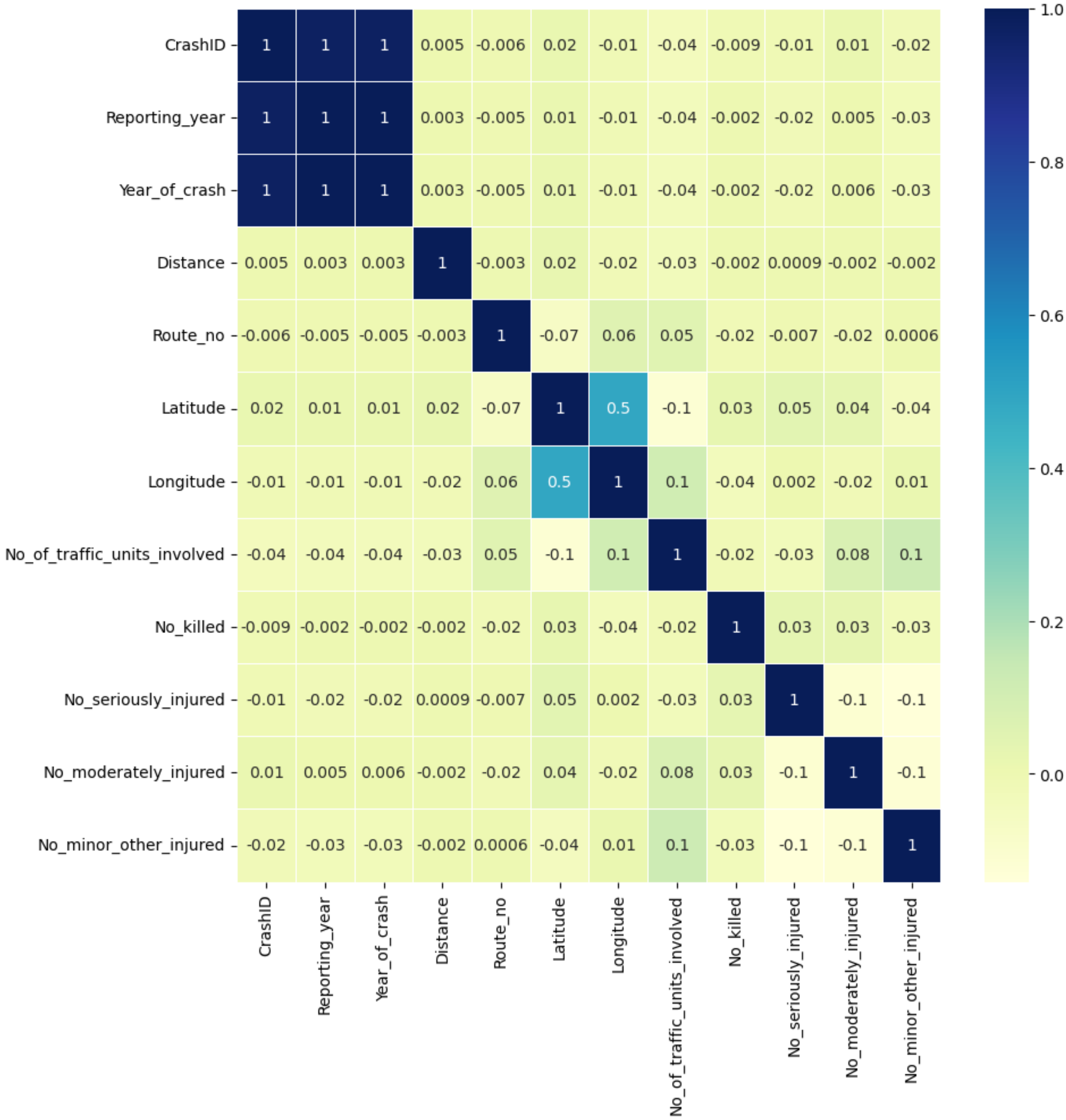
```
Note: you may need to restart the kernel to use updated packages.
ERROR: Invalid requirement: '#Install'
```

In [31]:
```
import plotly.express as px  #import plotly package
```

In [33]:
```
df7 = pd.DataFrame()
df7['Magnitude'] = df['No_killed'] + df['No_seriously_injured']
df7['Longitude'] = df['Longitude']
df7['Latitude'] = df['Latitude']
df7['Year_of_crash'] = df['Year_of_crash']

#Change data type for column year of crash
df7['Year_of_crash'] = df7['Year_of_crash'].astype(int)

#Clean out rows where the year is 2016 or 0
df7 = df7[(df7['Year_of_crash'] != 2016) & (df7['Year_of_crash'] != 0)]

#removes Magnitude where it is 0
df7 = df7[df7['Magnitude'] != 0]

#sets limits
df7['Magnitude'] = df7['Magnitude'].clip(1, 8)

#creating density heatmap
fig = px.density_mapbox(df7, lat='Latitude', lon='Longitude', z='Magnitude', radius=10, center=dict(lat=0, lon=180), zoom=0, mapbox_s
fig.show()
```

Heatmap to observe the correlation between variables

In [34]:
```
df_cleaned = df.drop(columns=['DCA_code', 'RUM_code'])
fig, ax = plt.subplots(figsize=(10,10))
cm = sns.heatmap(df_cleaned.corr(), linewidths = .5, cmap="YlGnBu", annot=True, ax=ax, fmt='.1g')
```



Predictive Modelling

Linear Regression - predict severity of crashes from speed limit

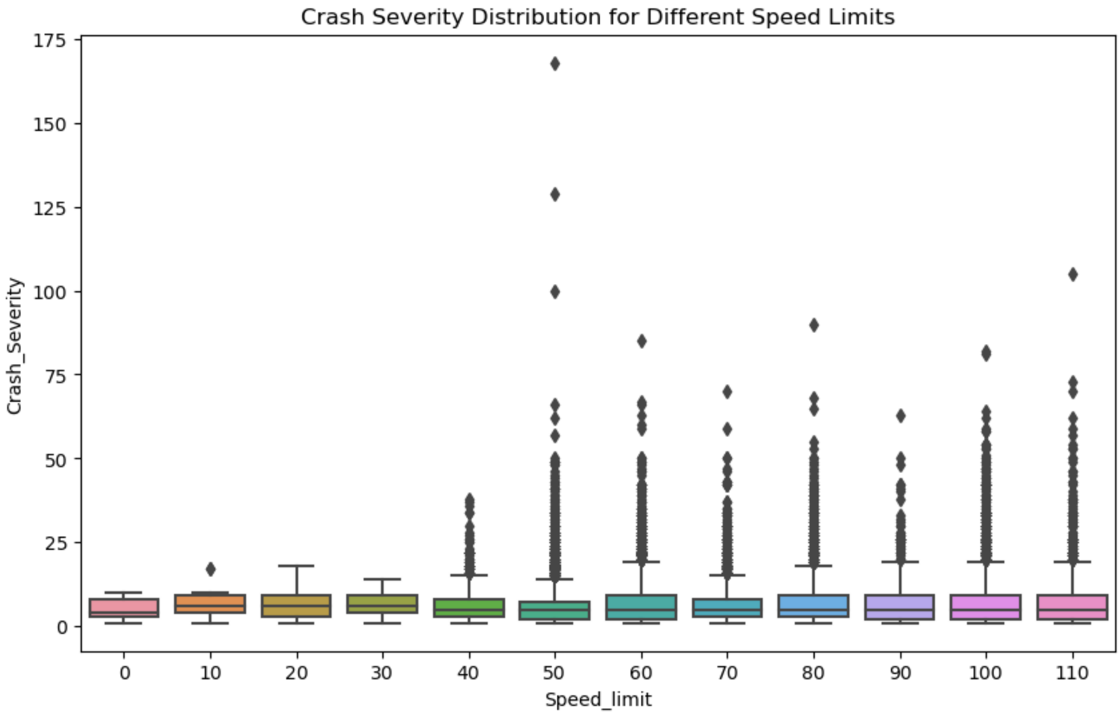
In [43]:
```
df.dtypes
```

Out[43]:
```
CrashID                            int64
Degree_of_crash                   object
Degree_of_crash_detailed          object
Reporting_year                     int32
Year_of_crash                      int32
Month_of_crash                    object
Day_of_week_of_crash              object
Two_hour_intervals                object
Street_of_crash                   object
Street_type                       object
Distance                         float64
Direction                         object
Identifying_feature               object
Identifying_feature_type          object
Town                              object
Route_no                         float64
School_zone_location              object
School_zone_active                object
Type_of_location                  object
Latitude                         float64
Longitude                        float64
LGA                               object
Urbanisation                      object
Alignment                         object
Primary_permanent_feature         object
Primary_temporary_feature         object
Primary_hazardous_feature         object
Street_lighting                   object
Road_surface                      object
Surface_condition                 object
Weather                           object
Natural_lighting                  object
Signals_operation                 object
Other_traffic_control             object
Speed_limit                     category
Road_classification               object
RUM_code                           int64
RUM_description                   object
DCA_code                           int64
DCA_description                   object
DCA_supplement                    object
First_impact_type                 object
Key_TU_type                       object
Other_TU_type                     object
No_of_traffic_units_involved       int64
No_killed                          int64
No_seriously_injured               int64
No_moderately_injured              int64
No_minor_other_injured             int64
dtype: object
```

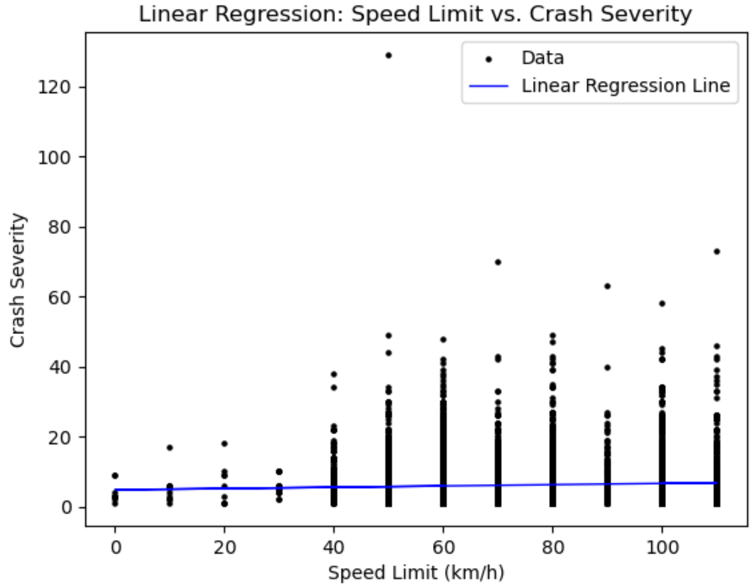
In [50]:
```
#Preparing data for modelling

#df['Year_of_crash'] = df['Year_of_crash'].fillna(0).astype(int)
#df['Reporting_year'] = df['Reporting_year'].fillna(0).astype(int)
df = df[(df['Year_of_crash'] != 2016) & (df['Year_of_crash'] != 0)]
df = df[(df['Reporting_year'] != 2016) & (df['Reporting_year'] != 0)]

df_data = df[['Speed_limit','No_of_traffic_units_involved', 'No_killed', 'No_seriously_injured', 'No_moderately_injured', 'No_minor_o

#df_data.columns[df_data.isna().any()].tolist()

df_data = df_data[df_data['Speed_limit'] != 'Unknown'].copy()
df_data['Speed_limit'].unique()

#df_data.head()

df_data['Speed_limit'] = df_data['Speed_limit'].str.extract('(\d+)').fillna(0).astype(int)

# Add the Crash_Severity column to the DataFrame
df_data['Crash_Severity'] = (
    df_data['No_killed'] * 16 +
    df_data['No_seriously_injured'] * 8 +
    df_data['No_moderately_injured'] * 4 +
    df_data['No_minor_other_injured'] * 2 +
    df_data['No_of_traffic_units_involved'] * 1
)
```

In [51]:
```
#Modelling Regression model

from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Extract features and target variable
X = df_data[['Speed_limit']]
y = df_data['Crash_Severity']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
```

```python
# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

plt.figure(figsize=(10, 6))
sns.boxplot(x='Speed_limit', y='Crash_Severity', data=df_data)
plt.title('Crash Severity Distribution for Different Speed Limits')
plt.show()

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

print(f'Coefficient: {model.coef_[0]}')
print(f'Intercept: {model.intercept_}')

import matplotlib.pyplot as plt

# Plot the data points
plt.scatter(X_test, y_test, color='black', label='Data', s=5)

# Plot the linear regression line
plt.plot(X_test, y_pred, color='blue', linewidth=1, label='Linear Regression Line')

# Labeling the plot
plt.title('Linear Regression: Speed Limit vs. Crash Severity')
plt.xlabel('Speed Limit (km/h)')
plt.ylabel('Crash Severity')
plt.legend()

# Show the plot
plt.show()
```




```
Mean Squared Error: 23.699084817856665
R-squared: 0.006749663582501952
Coefficient: 0.018437199801925434
Intercept: 4.755657655367552
```



Decision Tree - improve model; predict severity of crashes based on speed limit

In [53]:
```
#Preparing data for modelling decision tree

#convert year to integers
#df['Year_of_crash'] = df['Year_of_crash'].fillna(0).astype(int)
#df['Reporting_year'] = df['Reporting_year'].fillna(0).astype(int)

#exclude 2016
#df = df[(df['Year_of_crash'] != 2016) & (df['Year_of_crash'] != 0)]
#df = df[(df['Reporting_year'] != 2016) & (df['Reporting_year'] != 0)]


#Select columns
df_data = df[['Degree_of_crash_detailed','Speed_limit']].copy()

df_data.columns[df_data.isna().any()].tolist()

#Remove unknown from speed limit
df_data = df_data[df_data['Speed_limit'] != 'Unknown'].copy()
df_data['Speed_limit'].unique()

df_data.head()

#Convert string into integer, extracting number
df_data['Speed_limit'] = df_data['Speed_limit'].str.extract('(\d+)').fillna(0).astype(int)

df_data.head()
```

Out[53]:

	Degree_of_crash_detailed	Speed_limit
0	Fatal	50
1	Fatal	80
2	Fatal	100
3	Fatal	60
4	Fatal	100

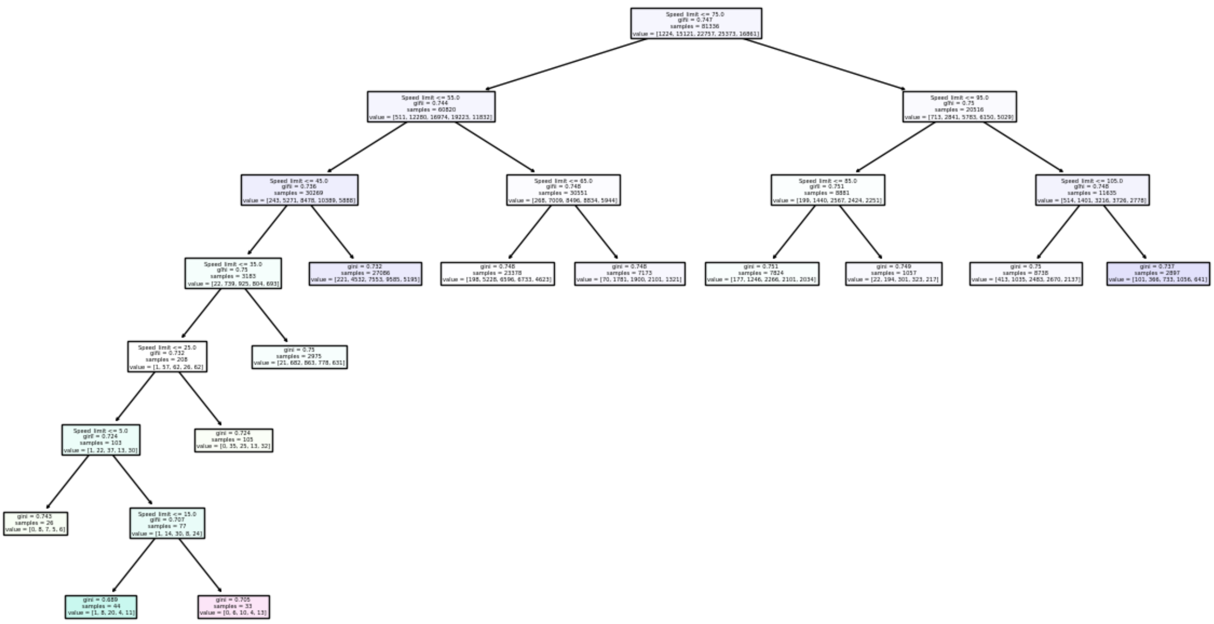
In [54]:
```
#Decision tree modelling

from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.tree import plot_tree
from sklearn.metrics import confusion_matrix

#Split data
X = df_data[['Speed_limit']]
y = df_data['Degree_of_crash_detailed']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model = DecisionTreeClassifier(random_state=42)

#Train model
model.fit(X_train, y_train)

#Make prediction
y_pred = model.predict(X_test)

#Print report
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')
```

```python
print('\nClassification Report:')
print(classification_report(y_test, y_pred))

#Visualise tree
plt.figure(figsize=(15, 8))
plot_tree(model, filled=True, feature_names=['Speed_limit'], class_names=None)
plt.show()
```

```
Accuracy: 0.3206294566019179

Classification Report:
```

```
C:\Users\maikh\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.

C:\Users\maikh\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.

C:\Users\maikh\anaconda3\lib\site-packages\sklearn\metrics\_classification.py:1318: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to cont
rol this behavior.
```

```
                        precision    recall  f1-score   support

                 Fatal       0.00      0.00      0.00       306
    Minor/Other Injury       0.22      0.00      0.00      3838
       Moderate Injury       0.30      0.15      0.20      5629
Non-casualty (towaway)       0.32      0.89      0.47      6353
        Serious Injury       0.40      0.00      0.00      4209

              accuracy                           0.32     20335
             macro avg       0.25      0.21      0.14     20335
          weighted avg       0.31      0.32      0.20     20335
```



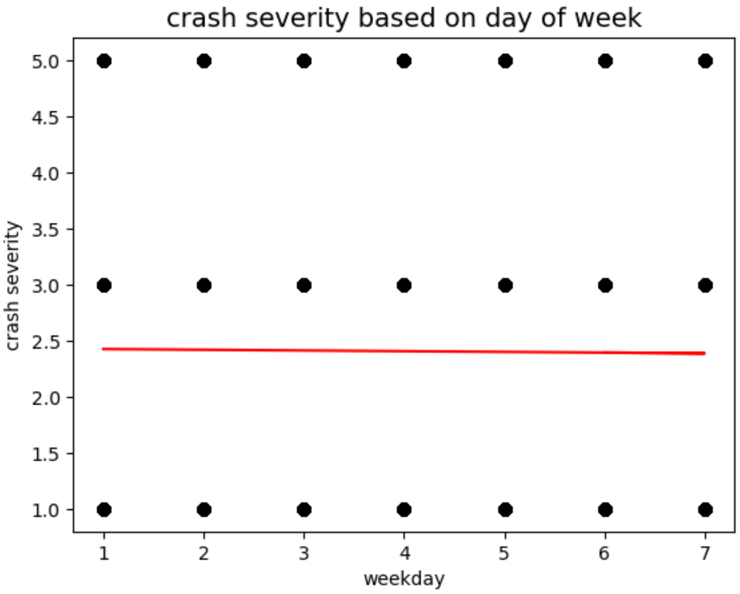
Linear Regression - Severity of crash vs. Day of week

In [57]:
```python
df_no_nan = df.dropna(subset=['Degree_of_crash', 'No_killed', 'Day_of_week_of_crash'])

from sklearn import linear_model

#clarifying each category into numerical values
def crash_severity(value):
    if value == "Fatal":
        return 5
    elif value == "Injury":
        return 3
    else:
        return 1
#assigning numerical values to days of week
def weekdayno(value):
    if value == "Monday":
        return 1
    elif value == "Tuesday":
        return 2
    elif value == "Wednesday":
        return 3
    elif value == "Thursday":
        return 4
```

```
    elif value == "Friday":
        return 5
    elif value == "Saturday":
        return 6
    elif value == "Sunday":
        return 7


df_no_nan['weekdayno'] = df_no_nan['Day_of_week_of_crash'].map(weekdayno)
df_no_nan['crash_severity'] = df_no_nan['Degree_of_crash'].map(crash_severity)


#df_no_nan.head()

regr = linear_model.LinearRegression()

severity= df_no_nan[['crash_severity']]
weekdayno= df_no_nan[['weekdayno']]

regr.fit(weekdayno, severity)

plt.scatter( df_no_nan.weekdayno, df_no_nan.crash_severity, color='Black')

plt.plot(weekdayno, regr.coef_[0][0]*weekdayno + regr.intercept_[0], '-r')
plt.xlabel("weekday")
plt.ylabel("crash severity")
plt.title('crash severity based on day of week', fontsize=14)
```

Out[57]:

```
Text(0.5, 1.0, 'crash severity based on day of week')
```



Logistic Regression - predict the likelihood of a crash being severe based on features like weather, road alignment, and road classification

In [75]:

```
#Import important libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
```

In [77]:

```
#Step 1 - Separate x and y data
from sklearn.preprocessing import LabelEncoder


X1 = df[['Weather', 'Alignment', 'Road_classification']]  #predictor variables
print(X1)

#Because the logistic reg only accepts categorical or numerical independent variables
X1 = pd.get_dummies(X1, columns=['Weather', 'Alignment', 'Road_classification'])

#Understanding severity of crashes
#road_accident['Degree_of_crash'].value_counts()

#There are 3 types of crash outcomes
#In order to conduct a logistic regression, these outcomes must be
#categorized into binary variables
#Decision: map fatal to value 1, Injury & Towaway to value 0


#Create binary variables to the crash severity variable
df['Crash_severe'] = df['Degree_of_crash'].map({'Fatal':1, 'Injury':1, 'Non-casualty (towaway)':0})
y1 = df[['Crash_severe']]
print(y1)

#Step 2 - Training and testing set split
```

```
X1_train, X1_test, y1_train, y1_test = train_test_split(
    X1, y1, test_size=0.2, random_state=42, stratify=y1
)


#After this, use resampling method: undersampling
```

```
         Weather Alignment Road_classification
0       Overcast    Curved               Local
1       Overcast    Curved               State
2           Fine  Straight               State
3       Overcast  Straight            Regional
4           Fine    Curved               State
...          ...       ...                 ...
101713      Fine    Curved               State
101714      Fine    Curved            Regional
101715  Overcast  Straight               Local
101716      Fine  Straight               State
101717      Fine  Straight               Local

[101718 rows x 3 columns]
        Crash_severe
0                  1
1                  1
2                  1
3                  1
4                  1
...              ...
101713             1
101714             1
101715             0
101716             0
101717             1

[101718 rows x 1 columns]
```

In [78]:

```
pip install -U imbalanced-learn   #Import package for undersampling
```

```
Note: you may need to restart the kernel to use updated packages.
ERROR: Invalid requirement: '#Import'
```

In [79]:

```
#Under-resampling technique

from imblearn.under_sampling import RandomUnderSampler # Import the necessary libraries

rus = RandomUnderSampler(random_state=42, sampling_strategy = 'majority')

# Balancing the data
X_resampled, y_resampled = rus.fit_resample(X1_train, y1_train)
```

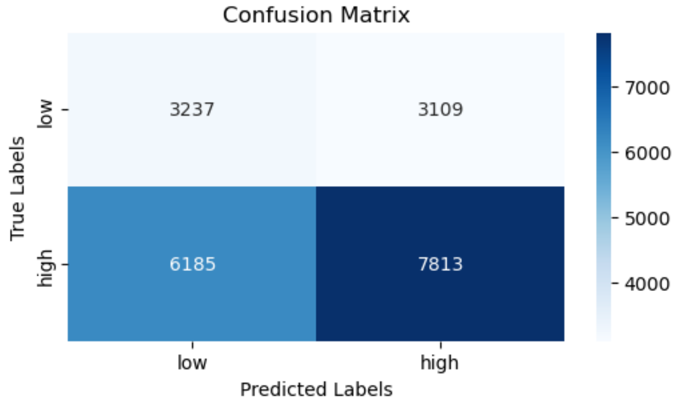
In [81]:

```
#Step 3 - build logistic regression model
r_accident_lr = LogisticRegression(solver = 'liblinear', random_state = 42, C = 10)

#Step 4 - Fit the model to the resampled data
r_accident_lr.fit(X_resampled, y_resampled)


#Step 5 - Predict the test data

y1_pred = r_accident_lr.predict(X1_test)



print(pd.DataFrame(y1_pred))

#Step 6 - Evaluate test the model
r_accident_lr.score(X1_test, y1_test)

#Evaluation using confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

def confusion_matrix_visual(y1_true, y1_pred, labels):
    cm = confusion_matrix(y1_true, y1_pred)

    plt.figure(figsize=(6, 3))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix')
    plt.show()

confusion_matrix_visual(y1_test, y1_pred, ['low', 'high'])

#Precision report
from sklearn.metrics import classification_report

print("Report : ",
    classification_report(y1_test, y1_pred))

# ROC Curves
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y1_test, r_accident_lr.predict_proba(X1_test)[:, 1])
```

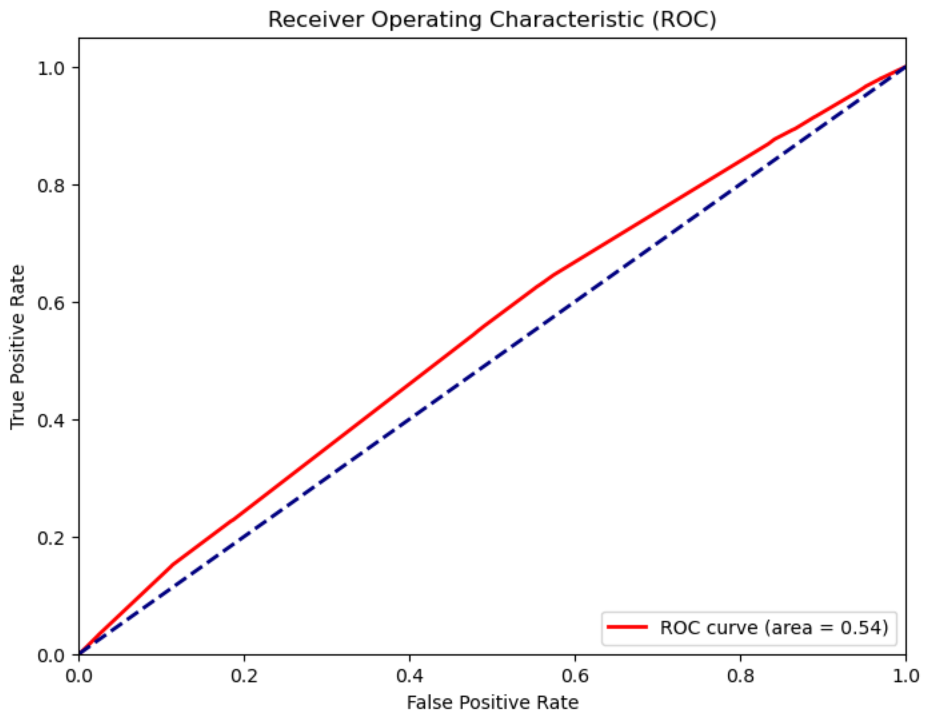
```
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

```
C:\Users\maikh\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
```

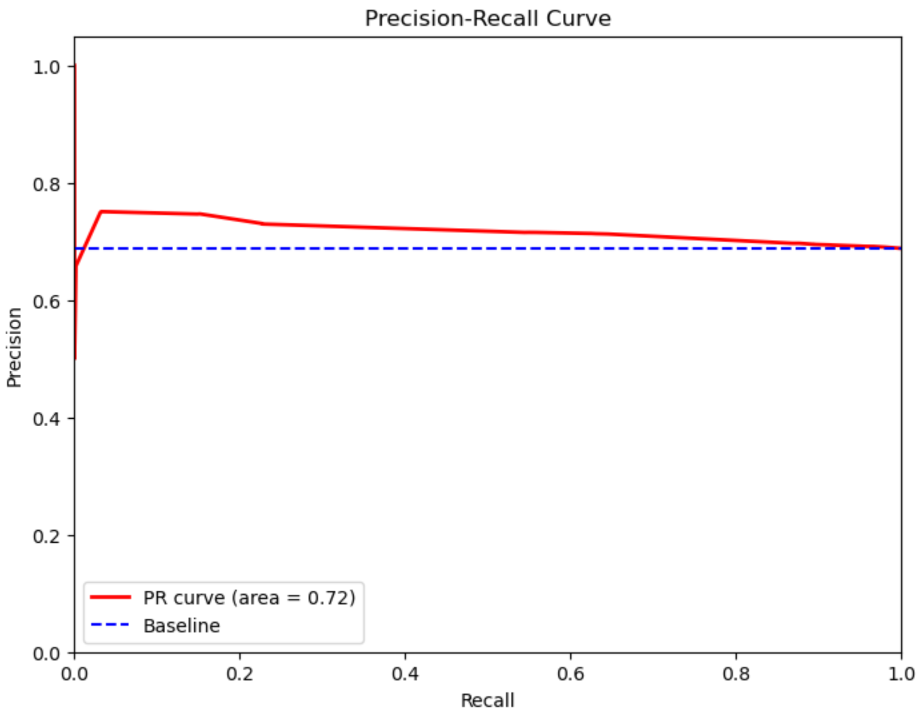
```
       0
0      1
1      1
2      1
3      0
4      0
...   ..
20339  0
20340  1
20341  1
20342  1
20343  1

[20344 rows x 1 columns]
```




```
Report :               precision    recall  f1-score   support

           0       0.34      0.51      0.41      6346
           1       0.72      0.56      0.63     13998

    accuracy                           0.54     20344
   macro avg       0.53      0.53      0.52     20344
weighted avg       0.60      0.54      0.56     20344
```



In [82]:
```
# Precision-Recall Curves
from sklearn.metrics import precision_recall_curve, auc
import matplotlib.pyplot as plt

# Calculate precision-recall curve
precision, recall, thresholds = precision_recall_curve(y1_test, r_accident_lr.predict_proba(X1_test)[:, 1])

# Calculate AUC for precision-recall curve
pr_auc = auc(recall, precision)

# Plot precision-recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='red', lw=2, label='PR curve (area = %0.2f)' % pr_auc)
plt.axhline(y=y1_test.mean().iloc[0], color='blue', linestyle='--', label='Baseline')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.legend(loc="lower left")
plt.show()
```



Feature selection for predictive models - using Chi-square method

In [89]:
```
import pandas as pd
from sklearn.feature_selection import chi2
from sklearn.preprocessing import LabelEncoder


# Convert categorical variables to numerical labels using LabelEncoder
label_encoder = LabelEncoder()
encoded_data = df.copy()
categorical_columns = ['Month_of_crash', 'Day_of_week_of_crash', 'Two_hour_intervals', 'Street_of_crash', 'Street_type', 'Direction',

for col in categorical_columns:
    encoded_data[col] = label_encoder.fit_transform(df[col])

# Create a DataFrame with only the relevant columns
X_categorical = encoded_data[categorical_columns]
y = encoded_data['Degree_of_crash']

# Apply chi-squared test
chi2_stat, p_values = chi2(X_categorical, y)

# Create a DataFrame to display the results
chi2_results = pd.DataFrame({
    'Feature': categorical_columns,
    'Chi2 Statistic': chi2_stat,
    'P-Value': p_values
})

# Display the results
print(chi2_results)
```

```
                    Feature  Chi2 Statistic        P-Value
0             Month_of_crash        9.010312   1.105187e-02
1        Day_of_week_of_crash       17.417811   1.651089e-04
2          Two_hour_intervals       27.484933   1.075778e-06
3             Street_of_crash    24857.818780   0.000000e+00
4                 Street_type      255.626863   3.099788e-56
5                   Direction        0.533835   7.657364e-01
6          Identifying_feature   102202.978255   0.000000e+00
7     Identifying_feature_type      835.101947  4.570000e-182
8                        Town      199.402831   5.014475e-44
9        School_zone_location       40.797710   1.383215e-09
10         School_zone_active        0.990990   6.092691e-01
11           Type_of_location     2063.758680   0.000000e+00
12                        LGA      969.804882  2.567708e-211
13               Urbanisation      695.568245  9.104889e-152
14                  Alignment       78.515515   8.924265e-18
15  Primary_permanent_feature      675.994832  1.620270e-147
16  Primary_temporary_feature        0.429538   8.067278e-01
17  Primary_hazardous_feature       26.674858   1.612977e-06
18            Street_lighting    12371.234424   0.000000e+00
19               Road_surface      129.416271   7.899759e-29
20          Surface_condition      683.618300  3.582385e-149
21                    Weather      715.145637  5.106202e-156
22           Natural_lighting      505.973206  1.346835e-110
23          Signals_operation      532.837324  1.976265e-116
24       Other_traffic_control       23.932638   6.354679e-06
25                Speed_limit      512.300052  5.694520e-112
26        Road_classification      103.361848   3.591367e-23
```

Logistic Regression - predict severity of crashes based on features with high chi2 stats: street of crash, type of location, street lighting

In [91]:

```
#Preparing data for modelling
X2 = df[['Street_of_crash', 'Street_lighting', 'Type_of_location']]
X2 = pd.get_dummies(X2, columns=['Street_of_crash', 'Street_lighting', 'Type_of_location'])


#Create binary variables to the crash severity variable
df['Crash_severe'] = df['Degree_of_crash'].map({'Fatal':1, 'Injury':1, 'Non-casualty (towaway)':0})
y2 = df[['Crash_severe']]
print(y2)


#Splitting training and testing dataset
from sklearn.model_selection import train_test_split # Import train_test_split function

X2_train, X2_test, y2_train, y2_test = train_test_split(X2, y2, test_size=0.3, random_state=1) # 70% training and 30% test
```

```
        Crash_severe
0                  1
1                  1
2                  1
3                  1
4                  1
...              ...
101713             1
101714             1
101715             0
101716             0
101717             1

[101718 rows x 1 columns]
```

In [92]:

```
pip install -U imbalanced-learn   #Import package for undersampling
```

```
Note: you may need to restart the kernel to use updated packages.
ERROR: Invalid requirement: '#Import'
```

In [93]:

```
from imblearn.under_sampling import RandomUnderSampler # Import the necessary libraries
rus = RandomUnderSampler(random_state=42, sampling_strategy = 'majority')
# Balancing the data
X2_resampled, y2_resampled = rus.fit_resample(X2_train, y2_train)
#Build model
r_accident_lr = LogisticRegression(solver = 'liblinear', random_state = 42, C = 10)
```

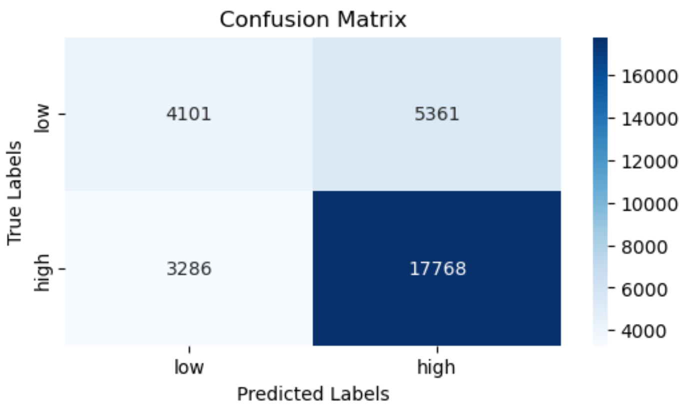
In [94]:

```
#Step 3: Fit the model with data

r_accident_lr.fit(X2_train, y2_train)


#Step 4: Predicting
y2_pred = r_accident_lr.predict(X2_test)

print(pd.DataFrame(y2_pred).value_counts())



#Step 5 - Evaluate test the model
accuracy = r_accident_lr.score(X2_test, y2_test)
print(accuracy)

#Confusion matrix
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

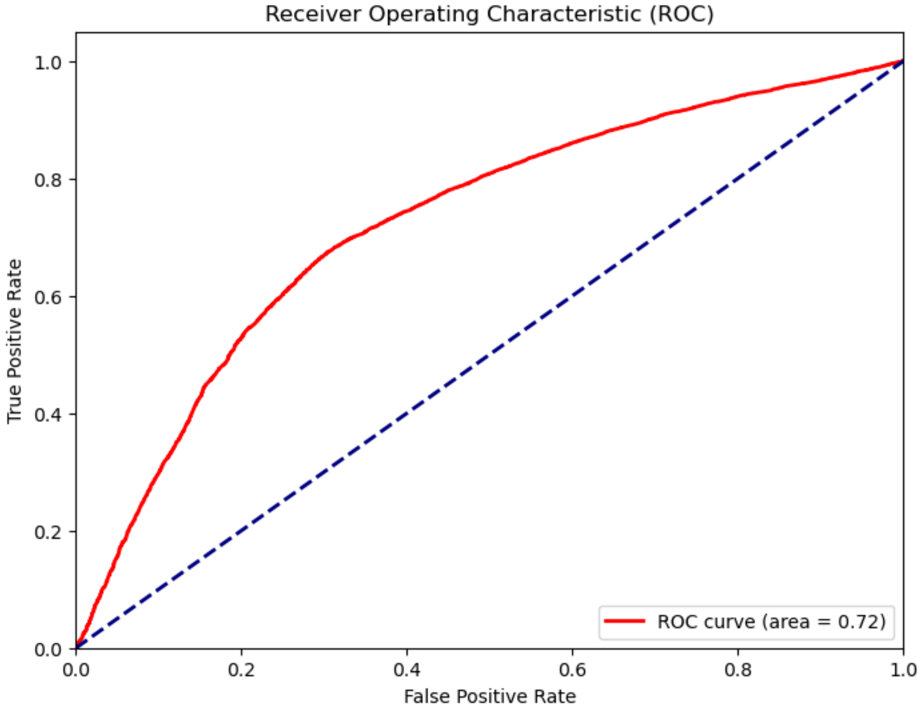
```python
def confusion_matrix_visual(y2_true, y2_pred, labels):
    cm = confusion_matrix(y2_true, y2_pred)

    plt.figure(figsize=(6, 3))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=labels, yticklabels=labels)
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix')
    plt.show()

confusion_matrix_visual(y2_test, y2_pred, ['low', 'high'])




#Precision report
from sklearn.metrics import classification_report

print("Report : ",
    classification_report(y2_test, y2_pred))

# ROC Curves
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y2_test, r_accident_lr.predict_proba(X2_test)[:, 1])
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='red', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc="lower right")
plt.show()
```

```
C:\Users\maikh\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning:

A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
```

```
1    23129
0     7387
dtype: int64
0.7166404509109975
```




```
Report :                precision    recall  f1-score   support

           0       0.56      0.43      0.49      9462
           1       0.77      0.84      0.80     21054

    accuracy                           0.72     30516
   macro avg       0.66      0.64      0.65     30516
weighted avg       0.70      0.72      0.71     30516
```


Receiver Operating Characteristic (ROC)
True Positive Rate
False Positive Rate
ROC curve (area = 0.72)
0.0
0.2
0.4
0.6
0.8
1.0