Reducing the number of high fatality accidents

📖 Background

You work for the road safety team within the department of transport and are looking into how they can reduce the number of major incidents. The safety team classes major incidents as fatal accidents involving 3+ casualties. They are trying to learn more about the characteristics of these major incidents so they can brainstorm interventions that could lower the number of deaths. They have asked for your assistance with answering a number of questions.

💾 The data

The reporting department have been collecting data on every accident that is reported. They've included this along with a lookup file for 2020's accidents. The lookup file further explain what values ineach column indicate.

Published by the department for transport. https://data.gov.uk/dataset/road-accidents-safety-data Contains public sector information licensed under the Open Government Licence v3.0.

> There are 27 columns in the accidents data and they are
>
> 1. **accident_index** : unique value for each accident. The accident_index combines the accident_year and accident_ref_no to form
> 2. **unique ID** : It can be used to join to Vehicle and Casualty
> 3. **accident_year** : year the accident occurred
> 4. **accident_reference** :In year id used by the police to reference a collision. It is not unique outside of the year, use accident index for linking to other years
> 5. longitude and latitude : indicate the location of the accident
> 6. accident_severity : This shows how serious the accident was. There are three categories -
>   - 1 - Fatal
>   - 2 - Serious
>   - 3 - Slight
> 7. number_of_vehicles : Number of vehicles involved the accident
> 8. number_of_casualties : Number of people injured or died from the accident
> 9. date : date of the accident
> 10. day_of_week : day the accident happened which is labeled 1-7 and represent sunday - staurday respectively
> 11. time : time of occurence. it is denoted as NaN if time is not known
> 12. first_road_class : describe the class of road in which the accident occurred. the following are the dneotion for the type of road -
>   - 1 : motorway - high speed roads that link major towns and cities
>   - 2 : A(M) - roads upgraded from major roads to motorway
>   - 3 : A - major roads between regional towns and cities
>   - 4 : B - minor roads. conect small town and villages
>   - 5 : C - classified unnumbered roads
>   - 6 : unclassified roads
> 13. first_road_number : these are numbers assighned to to various road class and they range from 1 - 9999 but their are exceptions -
>   - -1 - indicate an unknown road number
>   - 0 - first_road_class is C or Unclassified. These roads do not have official numbers so recorded as zero
> 14. road_type : specifies type of road. they have the following categories;
>   - 1 : roundabout
>   - 2 : one way street
>   - 3 : dual carriage way
>   - 6 : single carriage way
>   - 7 : slip road
>   - 9 : unknown
>   - 12 : one way street/slip road

- -1 : data missing or out of range

15. speed_limit : 20,30,40,50,60,70 are the only valid speed limits on public highways. there are some special speed limit;
    - -1 : speed is out of range or missing
    - 99 : speed is unknown or self-reported
16. junction_detail : information of the junction the accidents occurs and they can fall in the following categories;
    - 0 : not at junction or within 20 meters
    - 1 : roundabout
    - 2 : mini roundabout
    - 3 : T or staggered juction
    - 5 : slip roads
    - 6 : cross roads
    - 7 : More than 4 arms (not roundabout)
    - 8 : private drive or entrance
    - 9 : other junction
    - 99 : unknown or self-reported (it emans juction is known or someone reported it)
    - -1 : data missing or out of range
17. **junction_control** : indicate who or what is directiong traffic at the jusction at the time the accident occurred. it falls into the following categories;
    - 0 : nobody or nothing is present or they are within 20 meters from the junction
    - 1 : Authorised person
    - 2 : Auto-traffic signal
    - 3 : stop sign
    - 4 : Give way or uncontrolled
    - -1: data missing or out of range
    - 9 : unknown(self-reported)
18. **second_road_class** :
    - 0 : not at junction or within 20 meters
    - 1 : motorway - high speed roads that link major towns and cities
    - 2 : A(M) - roads upgraded from major roads to motorway
    - 3 : A - major roads between regional towns and cities
    - 4 : B - minor roads. conect small town and villages
    - 5 : C - classified unnumbered roads
    - 6 : unclassified roads
19. **second_road_number** : these are numbers assighned to to various road class and they range from 1 - 9999 but their are exceptions -
    - -1 - indicate an unknown road number
    - 0 - first_road_class is C or Unclassified. These roads do not have official numbers so recorded as zero
20. **pedestrian_crossing_human_control** : signifies who or what is contolling traffic at pedetrian crossing and it fall into the following categories
    - 0 : None within 50 meters
    - 1 : controlled by school crossing patrol
    - 2 : controlled by authorized person
    - -1: data missing or out of range
    - 9 : unknown or self-reported
21. **pedestrian_crossing_physical_facilities** : indicate any physical facilities that have been made available to ease pedestrian crossing. it has the following categories.
    - 0 : no physical crossing facilities within 50 meters
    - 1 : Zebra crossing
    - 4 : Pelican, puffin, toucan or similar non-junction pedestrian light crossing
    - 5 : Pedestrian phase at traffic signal junction
    - 7 : footbridge or subway
    - 8 : central refuge
    - -1 : data missing or out of range

- 9 : unknown or self-reported

22. **light_conditions**:
    - 1 : daylight
    - 4 : darkness - lights lit
    - 5 : darkness - lights unlit
    - 6 : darkness - no lighting
    - 7 : darkness - lighting unknown
    - -1: data missing or out of range
23. **weather_conditions** : condtion of the weather when the accident occur. It has the following categories
    - 1 : fine no high winds
    - 2 : raining no high winds
    - 3 : snowing no high winds
    - 4 : fine + high winds
    - 5 : raining + high winds
    - 6 : snowing + high winds
    - 7 - fog or mist
    - 8 : other
    - 9 : unknown
    - -1: data missing or out of range
24. **road_surface_conditions** : The condition of road surface. It has the following categories
    - 1 : dry
    - 2 : wet or damp
    - 3 : snow
    - 4 : frost or ice
    - 5 : flood over 3cm. deep
    - 6 : oil or diesel
    - 7 : mud
    - -1: data missing or out of range
    - 9 : unkown(self-reported)
25. **special_conditions_at_site** : it has the following categories
    - 0 : none
    - 1 : auto-traffic signal - out
    - 2 : auto signal part - defective
    - 3 : road sign or marking defective or obscured
    - 4 : roadworks
    - 5 : road surface defective
    - 6 : oil or diesel
    - 7 : mud
    - -1: data missing or out of range
    - 9 : unknown(self-reported)
26. **carriageway_hazards** : It has the following categories
    - 0 : None
    - 1 : vehicle load on the road
    - 2 : other object on the road
    - 3 : previous accident
    - 4 : dog on road
    - 5 : other animal on road
    - 6 : pedestrian in carriageway (not injured)
    - 7 : other animal in carriageway (except ridden horse)
    - -1: data missing or out of range
    - 9 : unknown(seld-reported)
27. **urban_or_rural_area** : environment where the accident happened. It has the following categories
    - 1 : urban
    - 2 : rural

> - 3 : unallocated
> - -1: data missing or out of range

💪 Competition challenge

Create a report that covers the following:

1. What time of day and day of the week do most major incidents happen?
2. Are there any patterns in the time of day/ day of the week when major incidents occur?
3. What characteristics stand out in major incidents compared with other accidents?
4. On what areas would you recommend the planning team focus their brainstorming efforts to reduce major incidents?

DATA WRANGLING

ASSESSMENT

> Import Necessary libraries

```
#install geopandas
!pip install geopandas
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting geopandas
  Downloading geopandas-0.10.2-py2.py3-none-any.whl (1.0 MB)
     |████████████████████████████████| 1.0 MB 13.6 MB/s
Requirement already satisfied: shapely>=1.6 in /usr/local/lib/python3.7/dist-packages (from geopandas) (1.8.5.post1)
Collecting fiona>=1.8
  Downloading Fiona-1.8.22-cp37-cp37m-manylinux2014_x86_64.whl (16.7 MB)
     |████████████████████████████████| 16.7 MB 707 kB/s
Collecting pyproj>=2.2.0
  Downloading pyproj-3.2.1-cp37-cp37m-manylinux2010_x86_64.whl (6.3 MB)
     |████████████████████████████████| 6.3 MB 35.9 MB/s
Requirement already satisfied: pandas>=0.25.0 in /usr/local/lib/python3.7/dist-packages (from geopandas) (1.3.5)
Collecting click-plugins>=1.0
  Downloading click_plugins-1.1.1-py2.py3-none-any.whl (7.5 kB)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas) (2022.9.24)
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas) (7.1.2)
Collecting cligj>=0.5
  Downloading cligj-0.7.2-py3-none-any.whl (7.1 kB)
Collecting munch
  Downloading munch-2.5.0-py2.py3-none-any.whl (10 kB)
Requirement already satisfied: six>=1.7 in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas) (1.15.0)
Requirement already satisfied: attrs>=17 in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas) (22.1.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from fiona>=1.8->geopandas) (57.4.0)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.25.0->geopandas) (2.8.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.25.0->geopandas) (1.21.6)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.25.0->geopandas) (2022.6)
Installing collected packages: munch, cligj, click-plugins, pyproj, fiona, geopandas
Successfully installed click-plugins-1.1.1 cligj-0.7.2 fiona-1.8.22 geopandas-0.10.2 munch-2.5.0 pyproj-3.2.1
```

```
# import libraries to be used
import warnings
warnings.filterwarnings('ignore')

#import sys
#!{sys.executable} -m pip install geopandas


from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# import packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
plt.style.use('seaborn-whitegrid')
from datetime import date
import descartes
import geopandas as gpd
from shapely.geometry import Point, Polygon
from sklearn.tree import DecisionTreeClassifier
from matplotlib.gridspec import GridSpec
```

```
%matplotlib inline
```

Data Wrangling

Major Steps

We need to understand the data and find out the real cause(s) of the accidents, so we will take the following two steps:

- Prepare the data for analysis
- Develop some visualization tools to help identify patterns in major incidents

Preparing the data

The data is first imported and a copy is made for each data to avoid tampering with the original data during the data wrangling phase.The data was checked for any missing entries. A few records seem to contain missing latitude and longitude entries. Since the number of incomplete records is small (14), and they do not belong to the class of interest (i.e., major incident), they can be safely dropped from the data, leaving a total of 91185 accidents to analyse. After ruling out the existence of duplicates records, the data is then passed through the following transformations:

- Date and time fields are converted to timestamps
- Month and Hour fields are extracted from said timestamps
- Major incidents are labelled 1, non-major 0
- Categorical features are identified, and their data type changed accordingly
- Lists of numerical and categorical features are prepared

```
# impport data
accidents = pd.read_csv('drive/MyDrive/Uk_accident_analysis/UK-accident-analysis/accident-data.csv')
display(accidents.head())
```

	accident_index	accident_year	accident_reference	longitude	latitude	accident_severity	number_of_vehicles	number_of_casualties
0	2020010219808	2020	10219808	-0.254001	51.462262	3	1	1
1	2020010220496	2020	10220496	-0.139253	51.470327	3	1	2
2	2020010228005	2020	10228005	-0.178719	51.529614	3	1	1
3	2020010228006	2020	10228006	-0.001683	51.541210	2	1	1
4	2020010228011	2020	10228011	-0.137592	51.515704	3	1	2

5 rows × 27 columns

```
# import lookup table (the table gives us more explanation about the data)
lookups = pd.read_csv('drive/MyDrive/Uk_accident_analysis/UK-accident-analysis/road-safety-lookups.csv')
```

```
# create a copy of the data
accident = accidents.copy()
lookup = lookups.copy()
```

```
# check for missing values and incorrect data types
accident.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 91199 entries, 0 to 91198
Data columns (total 27 columns):
 #   Column                                      Non-Null Count  Dtype
---  ------                                      --------------  -----
```

```
 0   accident_index                           91199 non-null  object 
 1   accident_year                            91199 non-null  int64  
 2   accident_reference                       91199 non-null  object 
 3   longitude                                91185 non-null  float64
 4   latitude                                 91185 non-null  float64
 5   accident_severity                        91199 non-null  int64  
 6   number_of_vehicles                       91199 non-null  int64  
 7   number_of_casualties                     91199 non-null  int64  
 8   date                                     91199 non-null  object 
 9   day_of_week                              91199 non-null  int64  
 10  time                                     91199 non-null  object 
 11  first_road_class                         91199 non-null  int64  
 12  first_road_number                        91199 non-null  int64  
 13  road_type                                91199 non-null  int64  
 14  speed_limit                              91199 non-null  int64  
 15  junction_detail                          91199 non-null  int64  
 16  junction_control                         91199 non-null  int64  
 17  second_road_class                        91199 non-null  int64  
 18  second_road_number                       91199 non-null  int64  
 19  pedestrian_crossing_human_control        91199 non-null  int64  
 20  pedestrian_crossing_physical_facilities  91199 non-null  int64  
 21  light_conditions                         91199 non-null  int64  
 22  weather_conditions                       91199 non-null  int64  
 23  road_surface_conditions                  91199 non-null  int64  
 24  special_conditions_at_site               91199 non-null  int64  
 25  carriageway_hazards                      91199 non-null  int64  
 26  urban_or_rural_area                      91199 non-null  int64  
dtypes: float64(2), int64(21), object(4)
memory usage: 18.8+ MB
```

```
# check records with null entries
accident[accident.isnull().any(axis = 1)]
```

	accident_index	accident_year	accident_reference	longitude	latitude	accident_severity	number_of_vehicles	number_of_casualti
25520	2020052002442	2020	052002442	NaN	NaN	2	2	
29452	2020070769852	2020	070769852	NaN	NaN	3	2	
32689	2020122001194	2020	122001194	NaN	NaN	3	2	
33578	2020137330369	2020	137330369	NaN	NaN	3	2	
81252	2020522005114	2020	522005114	NaN	NaN	3	3	
86437	2020622001016	2020	622001016	NaN	NaN	3	2	
86642	202063A017520	2020	63A017520	NaN	NaN	3	2	
86651	202063A018920	2020	63A018920	NaN	NaN	3	2	
86668	202063A025020	2020	63A025020	NaN	NaN	3	2	
86705	202063A035620	2020	63A035620	NaN	NaN	3	1	
86785	202063A059120	2020	63A059120	NaN	NaN	3	2	
87018	202063C020320	2020	63C020320	NaN	NaN	3	2	
87030	202063C024520	2020	63C024520	NaN	NaN	2	2	
87296	202063D061520	2020	63D061520	NaN	NaN	3	1	

14 rows × 27 columns

```
# drop rows with null entries
original_rows = accident.shape[0]
accident.dropna(inplace = True)
print('Dropped {} records with null entries'.format(original_rows- accident.shape[0]))
```

```
Dropped 14 records with null entries
```

```
# convert date and time to timestamp
accident['time_stamp'] = accident['date'] + ' ' + accidents['time']
accident['time_stamp'] = pd.to_datetime(accident['time_stamp'], format = '%d/%m/%Y %H:%M')

# verify
accident['time_stamp'].dtype
```

```
dtype('<M8[ns]')
```

```
# extract month and time from timestamp
accident['month'] = accident['time_stamp'].dt.month
accident['hour'] = accident['time_stamp'].dt.hour

#verify
accident[['month', 'hour']].head()
```

	month	hour
0	2	9
1	4	13
2	1	1
3	1	1
4	1	2

```
# list categorical features
categorical = list(accident.select_dtypes(include=['int64']).columns)
categorical.remove('number_of_vehicles')
categorical.remove('number_of_casualties')
```

```
# convert feature type to categorical
accident[categorical] = accident[categorical].astype('category')
```

```
# list all predictors per type
num_predictor = ['longitude', 'latitude', 'number_of_vahicles', 'number_of_casualties']
cat_predictor = ['day_of_week', 'first_road_class', 'road_type', 'speed_limit', \
                  'junction_detail', 'junction_control', 'second_road_class', \
                  'pedestrian_crossing_human_control', 'pedestrian_crossing_physical_facilities', \
                  'light_conditions', 'weather_conditions', 'road_surface_conditions', \
                  'special_conditions_at_site', 'carriageway_hazards', 'urban_or_rural_area', \
                  'month', 'hour']
```

```
# create a new column major incident
# that label accidents based on the number of
# casualties and severity level
accident['major_incident'] = [1 if(i==1)&(j>=3) else 0\
                              for i, j in zip(accident['accident_severity'],\
                                             accident['number_of_casualties'])]
```

```
# CREATE A LABEL FOR CATEGORIES
# drop records with NaN in the lookup table
label_1 = lookup.drop(['table', 'note'], axis = 1).dropna()
# create a months label
month_label = pd.DataFrame({'field_name' :['month' for i in range(1,13)],\
                            'code/format':[str(i) for i in range(1, 13)],\
                            'label':['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', \
                                    'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']})
# join the two label to create label
# for catgorical features
label = pd.concat([label_1, month_label], ignore_index = True, axis = 0)

# verify
label
```

	field name	code/format	label	field_name
0	accident_severity	1	Fatal	NaN
1	accident_severity	2	Serious	NaN
2	accident_severity	3	Slight	NaN

Developing data visualization tools

The following blocks of code show the forensic tools prepared to aid in our analysis for major incidents:

- Data subsets with major and non-major incidents
- Calculation of time between accidents
- Specific statistics for major and non-major incidents
- A function to plot said statistics as plain text
- A function to plot accident locations in the United Kingdom
- A function to plot average statistics per month
- A function to generate distribution of accidents and casualties per category
- A function to plot said distributions

```
# subset and create subgroups from the major_incident
# and separate into major and non major incident
major_incident = accident[accident['major_incident']==1]
minor_incident = accident[accident['major_incident']==0]


# calculate time between incidents
# calculate time bbetween major incidents
time_to_major = (major_incident['time_stamp'].iloc[-1] - \
                 major_incident['time_stamp'].iloc[0])/major_incident.shape[0]
time_to_major = time_to_major.seconds/3600 +24

# calculate time bbetween non-major incidents
time_to_non_major = (minor_incident['time_stamp'].iloc[-1] - \
                 minor_incident['time_stamp'].iloc[0])/minor_incident.shape[0]
time_to_non_major = time_to_non_major.seconds/60


# Create major incident values to plot in text form
major_val = []
major_val.append(str(np.round(major_incident.shape[0], 1)))
major_val.append(str(np.round(major_incident['number_of_casualties'].mean(),1)))
major_val.append(str(np.round(major_incident['number_of_vehicles'].mean(),1)))
major_val.append(str(np.round(time_to_major,1)))

# create major incident text
major_text = []
major_text.append('major_incident')
major_text.append('casualties pper \n major_incidents')
major_text.append('vehicles per \n major incident')
major_text.append('Hours between \n major incidents')


# Create major incident values to plot in text form
non_major_val = []
non_major_val.append(str(int(np.round(minor_incident.shape[0]/1000, 0)))+'k')
non_major_val.append(str(np.round(minor_incident['number_of_casualties'].mean(),1)))
non_major_val.append(str(np.round(minor_incident['number_of_vehicles'].mean(),1)))
non_major_val.append(str(np.round(time_to_non_major,1)))

# create major incident text
non_major_text = []
non_major_text.append('Non-major \n incidents')
non_major_text.append('casualties per \n non-major_incidents')
non_major_text.append('vehicles per \n non-major incident')
non_major_text.append('Hours between \n  non-major incidents')


# create function to plot the accidents statistics
def plot_stat(value, text, incident, ax):
  """
  This function plots the accidents the overall accident statistics
  as text with colors matching the incident type
  @ value : statistic value - type int
```

```
    @ text : the accompanying text explaining the value - type str
    @ incident : incident type - either major or non-major
    @ ax : the axis to plot each parameter
    """
    # define color for each category of incident
    if incident == 'major':
      color = 'darkorange'
    else:
      color = 'steelblue'

    # set up canvas
    _ = ax.set_aspect(0.8) # set aspect ratio i.e ratio of y_unit to x-unit

    # remove the spines (lines sorrounding the plot)
    for spine in ['top', 'bottom', 'left', 'right']:
      ax.spines[spine].set_visible(False)

    # set the tick labels
    _ = ax.set_xticklabels('')
    _ = ax.set_yticklabels('')

    # set axis limits
    ax.set(xlim =(0, 1), ylim =(0, 1))

    # set text annotations for each stat value
    _ = ax.text(0.5, 0.65, value, horizontalalignment = 'center',\
                verticalalignment = 'center', fontsize = 55,\
                fontweight = 'semibold', color = color)
    _ = ax.text(0.5, 0.3, text, horizontalalignment = 'center',\
                verticalalignment = 'center', fontsize = 15,\
                fontweight = 'demibold', color = color)

    # set the tight layout
    _ = plt.tight_layout()


  # create a function to plot map and accident locations
  def plot_map(filename, df, ax, incident = None, text= ''):
    '''
    This function plots map of UK and the location of accident occurences
    @ filename : name of file to plot map of UK
    @ df : dataframe that contains longitude and latitude of accident locations
    @ ax : axes of plotting
    @ text
    '''
    # read file containing cordinates of UK
    uk_map = gpd.read_file(filename)
    # set the cordinate referencing system
    crs = {'init': 'epsg:4326'}
    # set the cordinates
    geometry = [Point(xy) for xy in zip(df['longitude'], df['latitude'])]
    # create a geodataframe using the above parameters
    geo_df = gpd.GeoDataFrame(df, crs = crs, geometry = geometry)

    # plot map
    _ = uk_map.plot(ax= ax, alpha = 0.6, color = 'grey')

    # plot accident locations
    if(incident == 'non_major'):
      _ = geo_df[geo_df['major_incident'] == 0].plot(ax =ax, markersize = 7,\
                                                    color= 'steelblue', marker = 'o',\
                                                    label = 'non-major incident')
    if(incident == 'major'):
      _ = geo_df[geo_df['major_incident'] == 1].plot(ax =ax, markersize = 20,\
                                                    color= 'darkorange', marker = 'o',\
                                                    label = 'major incident')
    if(incident != 'non_major') & (incident != 'major'):
      _ = geo_df[geo_df['major_incident'] == 0].plot(ax =ax, markersize = 7,\
                                                    color= 'steelblue', marker = 'o',\
                                                    label = 'non-major incident')
      _ = geo_df[geo_df['major_incident'] == 1].plot(ax =ax, markersize = 20,\
                                                    color= 'darkorange', marker = 'o',\
                                                    label = 'major incident')
      # set title, axis labels and limits
      _ = ax.set_title('Accident locations' +text+ '\n\n', fontsize=18)
      _ = ax.set_xlabel('Longitude', fontsize=15)
      _ = ax.set_ylabel('Latitude', fontsize=15)
```

```
    _ = ax.set_xlim(xmin=-10)
    _ = ax.set_ylim(ymax=61)
    _ = ax.legend(loc = 'upper center', frameon = True,\
                  fontsize = 'x-large', bbox_to_anchor = (0.5, 1.07),\
                  ncol = 2)
    _ = plt.tight_layout()


# Create a function to plot average statistics per month
def plot_monthly_avg(df, col, incident, ax):
  '''
  plots monthly average statistics
  @ df : dataframe of interest
  @ col : column of interest in the data
  @ incident : category of the incident - type str
  @ ax : axes of the subplots
  '''
  # define color and title for each incident type
  if incident == 'major':
    color = 'darkorange'
    _ = ax.set_title('Average '+col+ ' in major accidents',\
                     fontsize = 18)
  else:
    color = 'steelblue'
    _ = ax.set_title('Average '+col+ ' in non-major accidents',\
                     fontsize = 18)

  # group by month and date
  group = df.groupby(['month', 'date'], as_index = False)[col].agg(['size', 'mean'])
  # plot data
  _ = sns.lineplot(data = group, x = 'month', y = 'mean',
                   ax = ax, ci = 95,\
                   color = color)
  # set title, axis labels and layout
  _ = ax.set_xlim(xmin = 1, xmax = 12)
  _ = ax.set_xlabel('Months', fontsize=16)
  _ = ax.set_ylabel('Average', fontsize=16)
  _ = ax.set_xticks([i for i in range(1, 13)])
  _ = ax.tick_params(which = 'both', labelsize = 14)
  _ = plt.tight_layout()



# create a function to generate distribution of
# accidents and casualties per categorical feature
def cat_distribution(col, df = accident):
    """Function that generates distribution of accidents and casualties
    per incident type in given categorical column and returns grouped data
    @ col : columns or field in the data
    @ df : dataframe of interest
    """
    # Group categorical feature with numbers of accidents and casualties for each
    # incident type
    part1 = df.groupby(['major_incident', col], as_index = False)\
                      ['number_of_casualties'].size()
    part2 = df.groupby(['major_incident', col], as_index = False)\
                      ['number_of_casualties'].sum()
    group = pd.concat([part1, part2[['number_of_casualties']]], axis = 1)

        # Calculate percentage of accidents per category for each incident type
    frac_0 = group[group['major_incident']==0]['size']/ \
             group[group['major_incident']==0]['size'].sum()
    frac_1 = group[group['major_incident']==1]['size']/ \
             group[group['major_incident']==1]['size'].sum()
    group['fraction'] = 100*pd.concat([frac_0, frac_1], ignore_index=True, \
                                      axis=0).round(4)

    # Calculate percentage of casualties per category for each incident type
    cas_0 = group[group['major_incident']==0]['number_of_casualties']/ \
            group[group['major_incident']==0]['number_of_casualties'].sum()
    cas_1 = group[group['major_incident']==1]['number_of_casualties']/ \
            group[group['major_incident']==1]['number_of_casualties'].sum()
    group['perc_casualties'] = np.round(100*pd.concat([cas_0, cas_1], \
                                                      ignore_index=True, \
                                                      axis=0), 2)

    # Label categories within group
```

```
    group['field name'] = col
    group[col] = group[col].astype('string')
    group = group.merge(label, how='left', left_on=['field name', col], \
                        right_on=['field name', 'code/format'])
    group[col] = group[col].astype('int64')

    # Fill any null labels with appropriate values
    if group['label'].isnull().any():
        group['code/format'] = group[col].astype('string')
        group['label'] = group[col].astype('string')

    return group


# Create function to plot distribution of accidents and casualties in categorical feature
def plot_bullets(col, ax1, ax2, df=accident, text=''):
    """Function that plots distribution of accidents
    and casualties per incident type in given list of
    categorical columns"""

    # Create group to plot
    group = cat_distribution(col, df)

    # Code to sort values according to categorical feature used
    #if (col=='hour') | (col=='hour') | (col=='day_of_week'):
    #    group = group.sort_values(['major_incident', col], ascending=[True, False])

    # Create the Bar chart

    _ = ax1.barh(data=group[group['major_incident']==0], width='fraction', y='label', \
                 color='steelblue', label='Non-major_incident', height=0.7)
    _ = ax1.barh(data=group[group['major_incident']==1], width='fraction', y='label', \
                 color='darkorange', label='Major_incident', height=0.2)
    _ = ax1.set_xlabel('% accidents within class', fontsize=16)
    _ = ax1.tick_params(which='both', labelsize=14)
    _ = ax1.set_title('Accident distribution per '+col+text+'\n\n', fontsize=18)
    _ = ax1.legend(loc='upper center', frameon=True, fontsize='large', \
                   bbox_to_anchor=(0.5, 1.06), ncol=2)

    # create plot for Percentage casualties
    x = [1 for i in range(len(group[group['major_incident']==1][col]))]
    _ = sns.scatterplot(data=group[group['major_incident']==1], x=x, y='label', \
                        palette='Oranges', label='% total\ncasualties \nin major \nincidents', \
                        size='perc_casualties', sizes=(20,500), ax=ax2, hue='perc_casualties')
    _ = ax2.set_xlabel('')
    _ = ax2.set_ylabel('')
    _ = ax2.legend(loc='upper left', bbox_to_anchor=(1.01, 1),borderaxespad=0, \
                   frameon=True, fontsize=12)
    _ = ax2.tick_params(labelbottom=False, labelleft=False)
```

Data Exploration

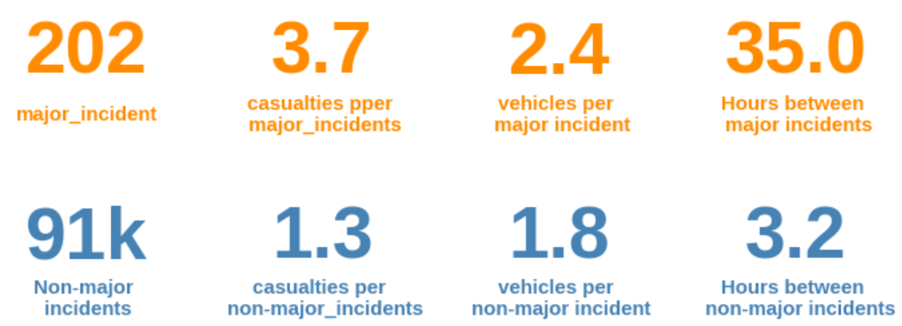
We have taken the first step of separating the accident occurences into tow categories (minor and major). All we need to do now is to begin our analysis using these labels. we can continue exploring how major incidents differ from all other accidents combined. The visualization tools developed in the previous section will help us make these comparisons between major and non-major incidents.

Major Incidents are three times more deadlier than any other incident

We can already see the disparity between major incidents and all other incidents. Major incidents are comparatively rarer than al other incidents occuring roughly once every other day while others occur about 8 times everyday. They represent a tiny fraction (1%) of the 91,185 accidents recorded in 202. The casualties involved in major incidents are about three times more thann in other incidents and there. This means that on average, major incidents kill 4 people every other day. Major incidents involve 30% more vehicles than any other incidents

```
# plot text value of the accidents statistics
plt.style.use('seaborn-white')

# create subplot
fig, ax = plt.subplots(2, 4, figsize = (10, 4))
```

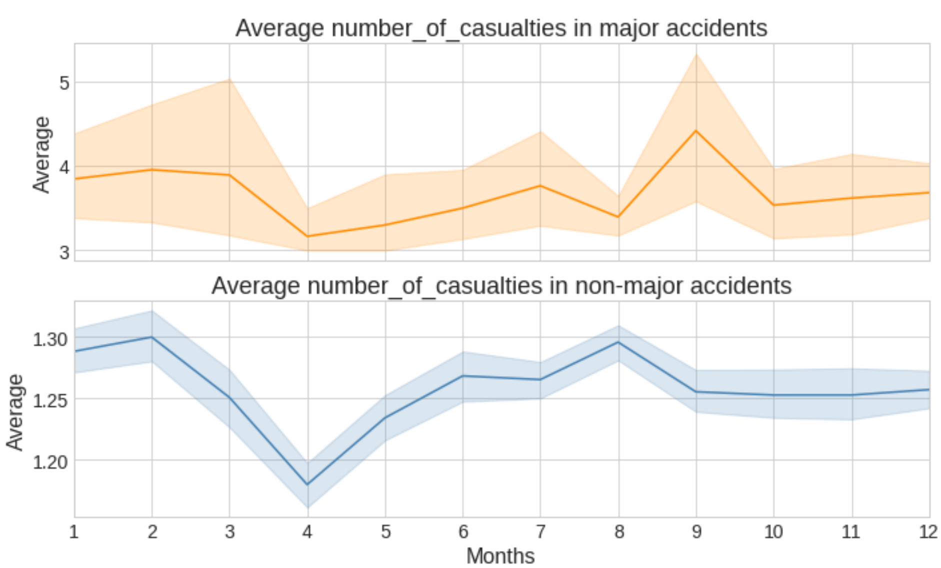
```
ax = ax.ravel()
m = 0

# plot
for value, text in zip(major_val, major_text):
  plot_stat(value, text, 'major', ax[m])
  m += 1
for value, text in zip(non_major_val, non_major_text):
  plot_stat(value, text, 'Non-major', ax[m])
  m += 1
```



The average number of vehicles and casualties varies every month. Looking at the plot of the averages with a 95% confidence interval (95% confidence interval is represented by the shaded region), we can get a sense of how the averages would vary if there were to be accidents again many times.

Comparing the average casulaties, there is a high level of variability for the major incidents and we can see that for the major incident, at least 3 people die whenever it occurs and we see the number increasing to above 4 between August and september. We can expect the number to increase to 5 if the incidents were to occur many times again.

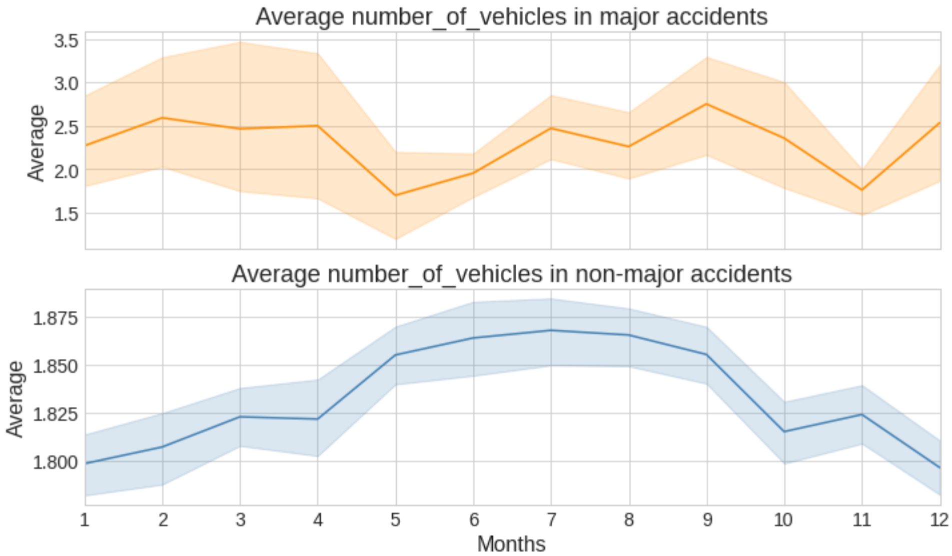
```
# plot average casualties per month for each incident type
fig, ax = plt.subplots(2,1, figsize = (10,6), sharex = True)

plot_monthly_avg(major_incident, 'number_of_casualties', 'major', ax[0])
plot_monthly_avg(minor_incident, 'number_of_casualties', 'non-major', ax[1])
```



Double-click (or enter) to edit

```
# plot average average vehicles per month for each incident type
fig, ax = plt.subplots(2,1, figsize = (10,6), sharex = True)
```

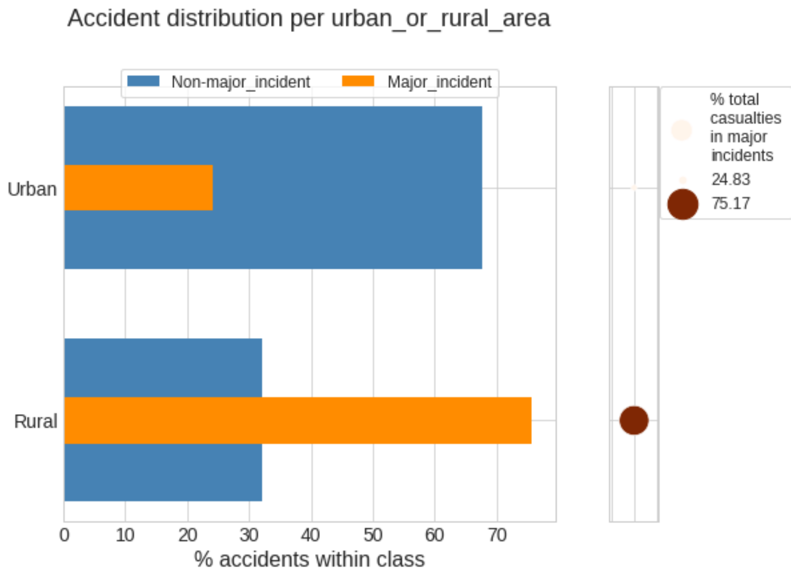
```
plot_monthly_avg(major_incident, 'number_of_vehicles', 'major', ax[0])
plot_monthly_avg(minor_incident, 'number_of_vehicles', 'non-major', ax[1])
```



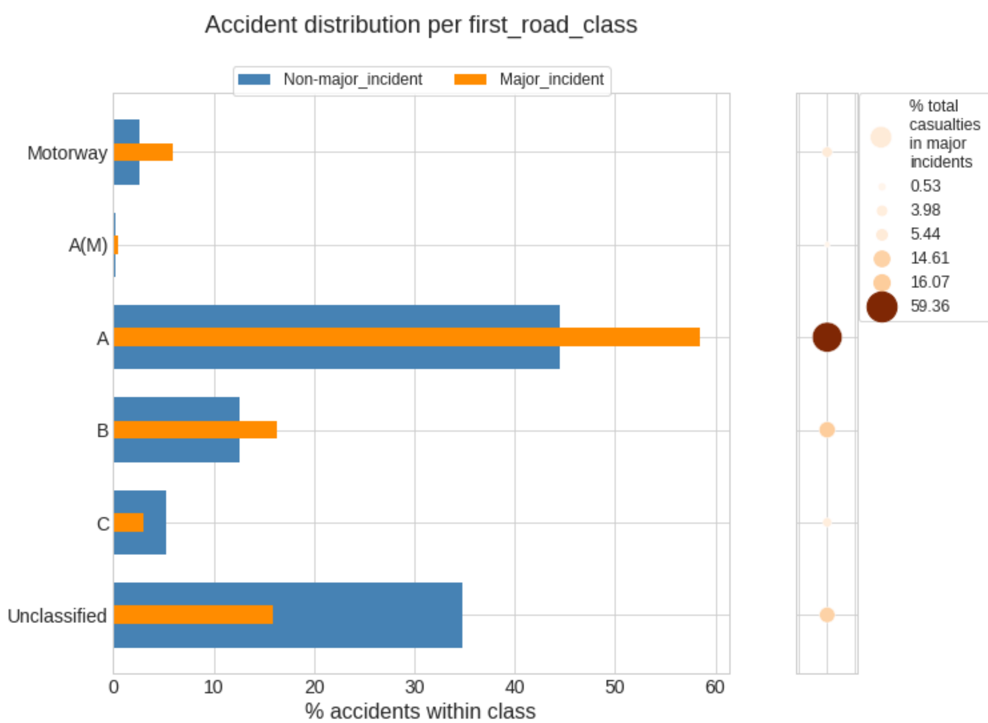
Major Incidents frequent single carriageways in rural areas

```
# plot map location of accidents
plt.style.use('seaborn-whitegrid')
fig, ax = plt.subplots(figsize = (10,10))

plot_map(r'drive/MyDrive/Uk_accident_analysis/UK-accident-analysis/GBR_adm2.shp', accident, ax)
```

```
# Plot distribution of accidents and casualties per urban or rural areas
#plt.style.use('seaborn-whitegrid')
fig, ax = plt.subplots(1 ,2, figsize= (8, 6),
                       sharey = True, gridspec_kw= dict(width_ratios = [3, 0.3]))
plot_bullets('urban_or_rural_area', ax[0], ax[1])
```

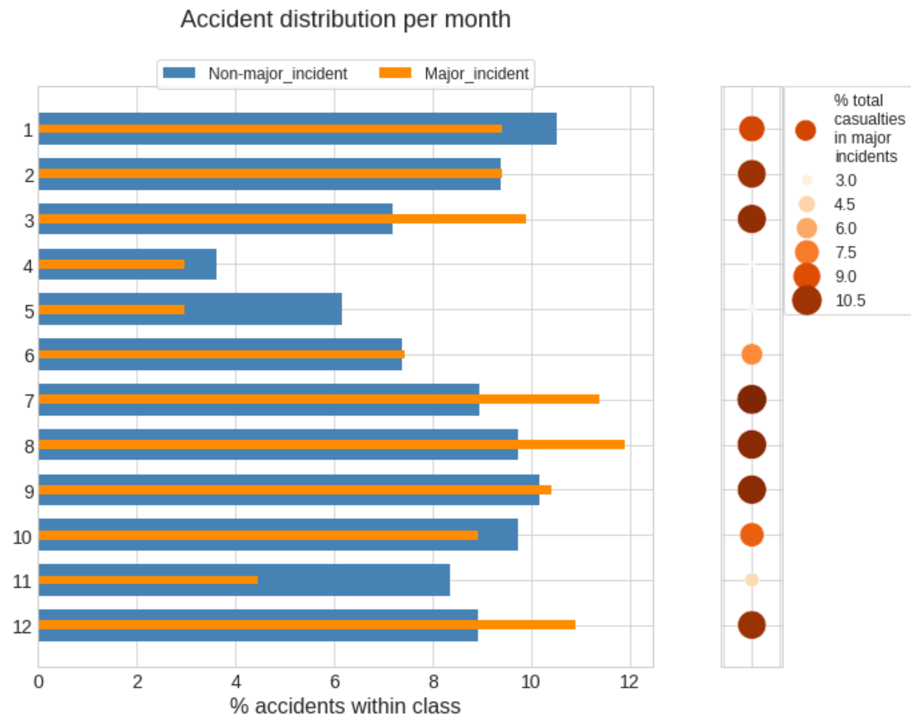



```
# Plot distribution of accidents and casualties per road class

fig, ax = plt.subplots(1 ,2, figsize= (10, 8),
                       sharey = True, gridspec_kw= dict(width_ratios = [3, 0.3]))
plot_bullets('first_road_class', ax[0], ax[1])
```

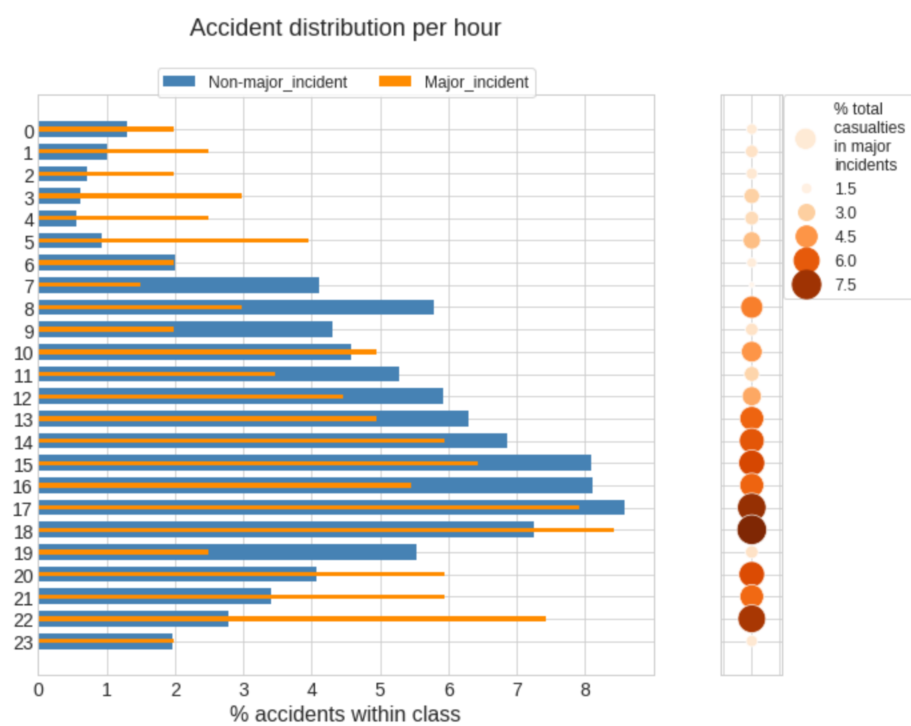


Major incidents peak in late summer, weekends, and late afternoons

```
# plot the distribution of accidents and caulaties for each month
fig, ax = plt.subplots(1, 2, figsize = (10, 8),\
                       sharey = True, gridspec_kw = dict(width_ratios = [3, 0.3]))
plot_bullets('month', ax[0], ax[1])
```




```
# plot the distribution of accidents and caulaties for per hour
fig, ax = plt.subplots(1, 2, figsize = (10, 8),\
                       sharey = True, gridspec_kw = dict(width_ratios = [3, 0.3]))
plot_bullets('hour', ax[0], ax[1])
```

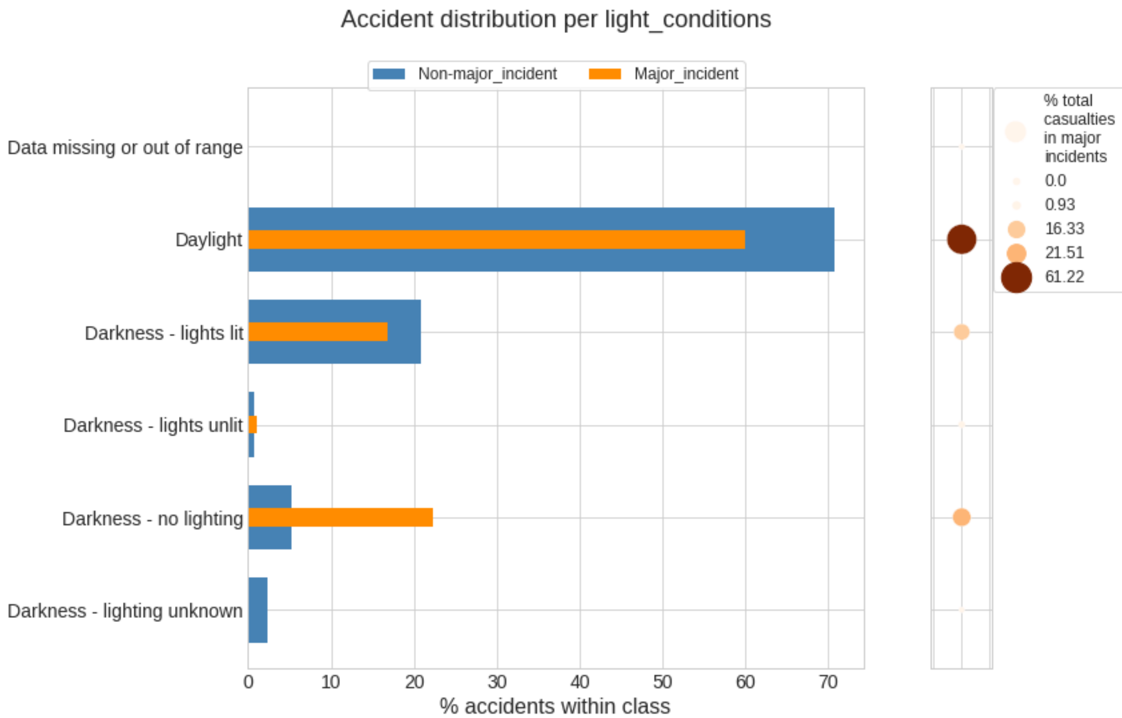



```
cas_0 = accident[accident['major_incident']==0]['number_of_casualties']/ \
            accident[accident['major_incident']==0]['number_of_casualties'].sum()
```

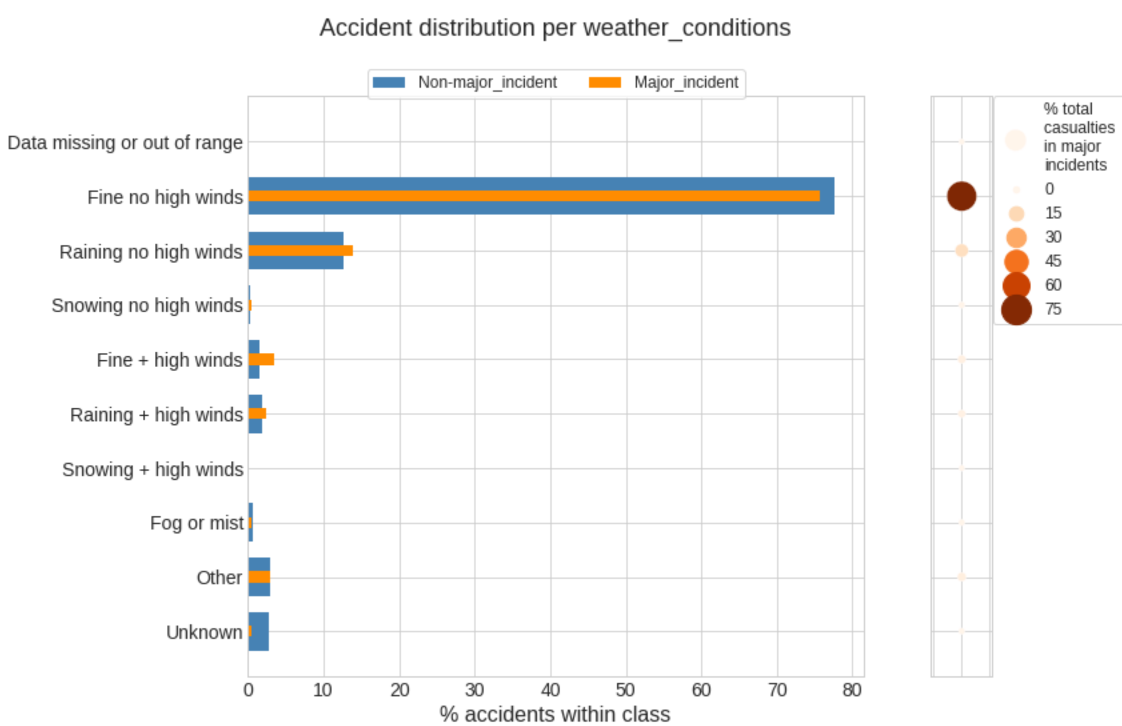
Major incidents strike in broad daylight, fine weather, and at 60-mph limits

```
# plot the distribution of accidents and caulaties per
# light conditions
fig, ax = plt.subplots(1, 2, figsize = (10, 8),\
```

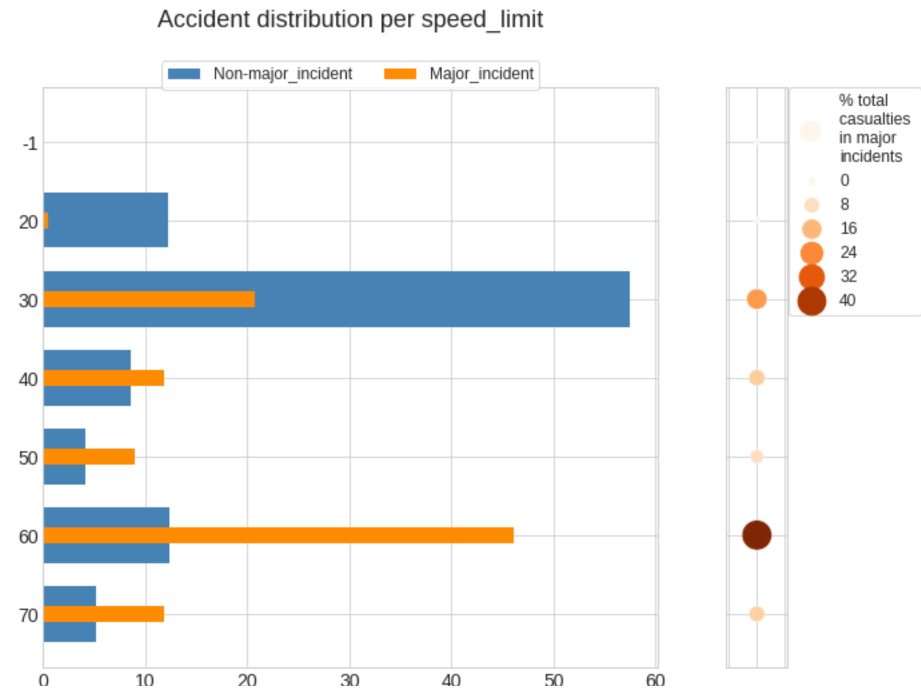
```
                      sharey = True, gridspec_kw = dict(width_ratios = [3, 0.3]))
plot_bullets('light_conditions', ax[0], ax[1])
```



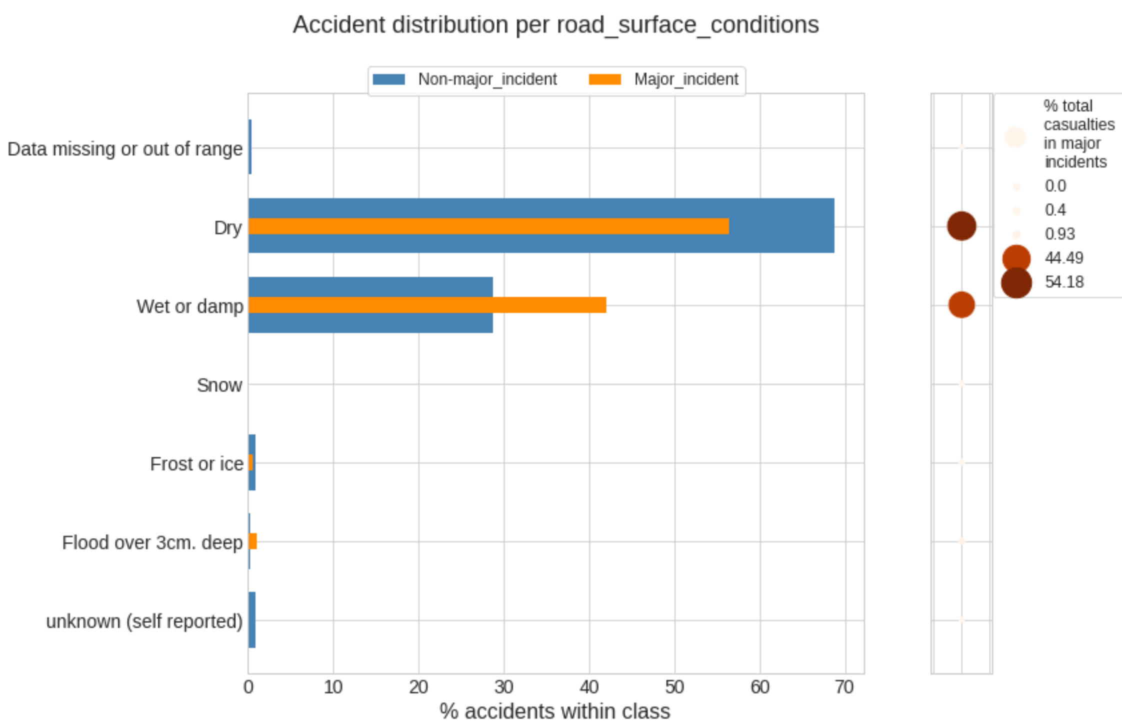

```
# plot the distribution of accidents and caulaties per
# weather condition
fig, ax = plt.subplots(1, 2, figsize = (10, 8),\
                      sharey = True, gridspec_kw = dict(width_ratios = [3, 0.3]))
plot_bullets('weather_conditions', ax[0], ax[1])
```




```
# plot the distribution of accidents and caulaties per
# light conditions
fig, ax = plt.subplots(1, 2, figsize = (10, 8),\
                      sharey = True, gridspec_kw = dict(width_ratios = [3, 0.3]))
plot_bullets('speed_limit', ax[0], ax[1])
```




```
# plot the distribution of accidents and caulaties per
# light conditions
fig, ax = plt.subplots(1, 2, figsize = (10, 8),\
                       sharey = True, gridspec_kw = dict(width_ratios = [3, 0.3]))
plot_bullets('road_surface_conditions', ax[0], ax[1])
```

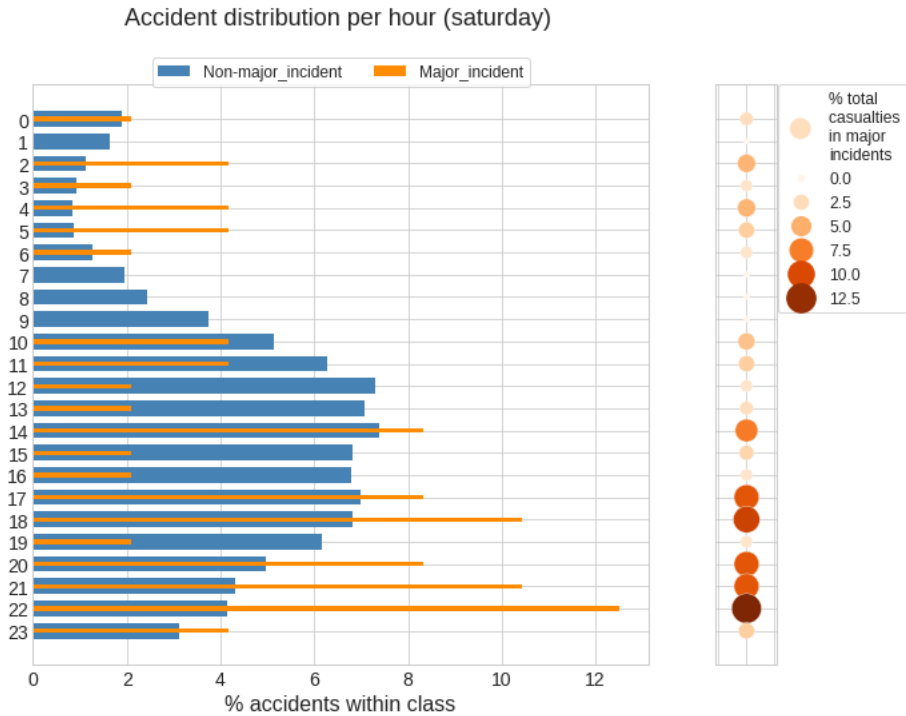


Finding the deadliest hour

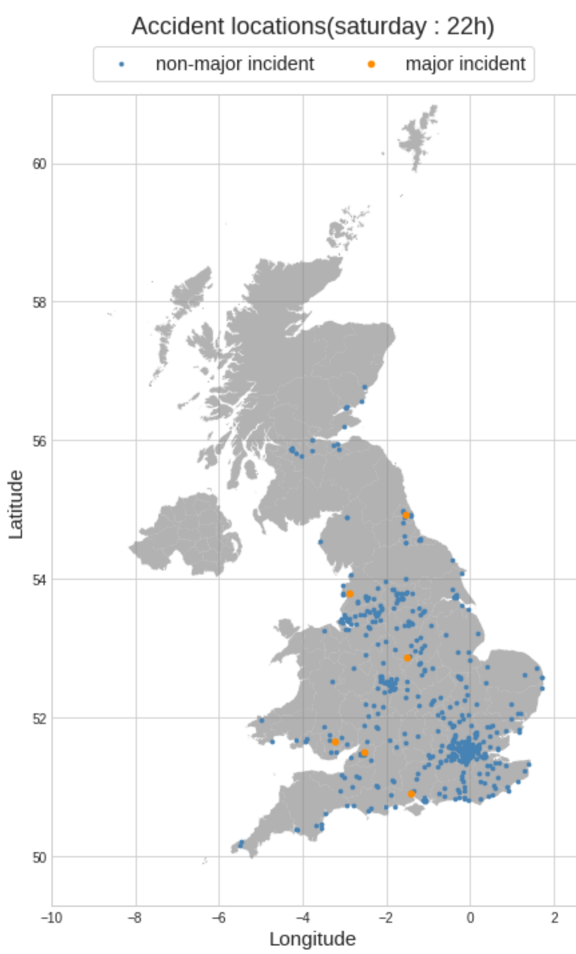
Saturday at 10pm is the deadliest time

```
# plot the distributon of accident and casulaties per hour
# on saturdays
fig, ax = plt.subplots(1, 2, figsize = (10, 8),\
                       sharey = True, gridspec_kw = dict(width_ratios = [3, 0.3]))
```

```
saturday = accident[accident['day_of_week']==7]
plot_bullets('hour', ax[0], ax[1], saturday, text = ' (saturday) ')
```

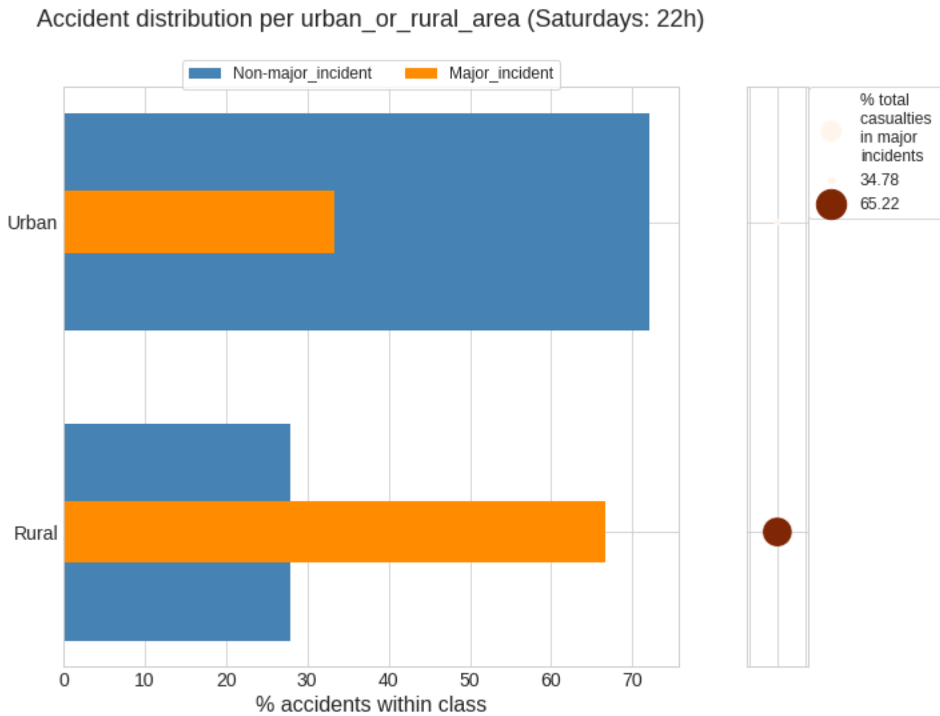



```
fig, ax = plt.subplots(figsize = (10,10))
saturday_22h = accident[(accident['day_of_week']==7) & (accident['hour'] ==22)]
plot_map(r'drive/MyDrive/Uk_accident_analysis/UK-accident-analysis/GBR_adm2.shp', saturday_22h, ax, text = '(saturday : 22h)')
```



Major incidents at the deadliest time are more influenced by external conditions

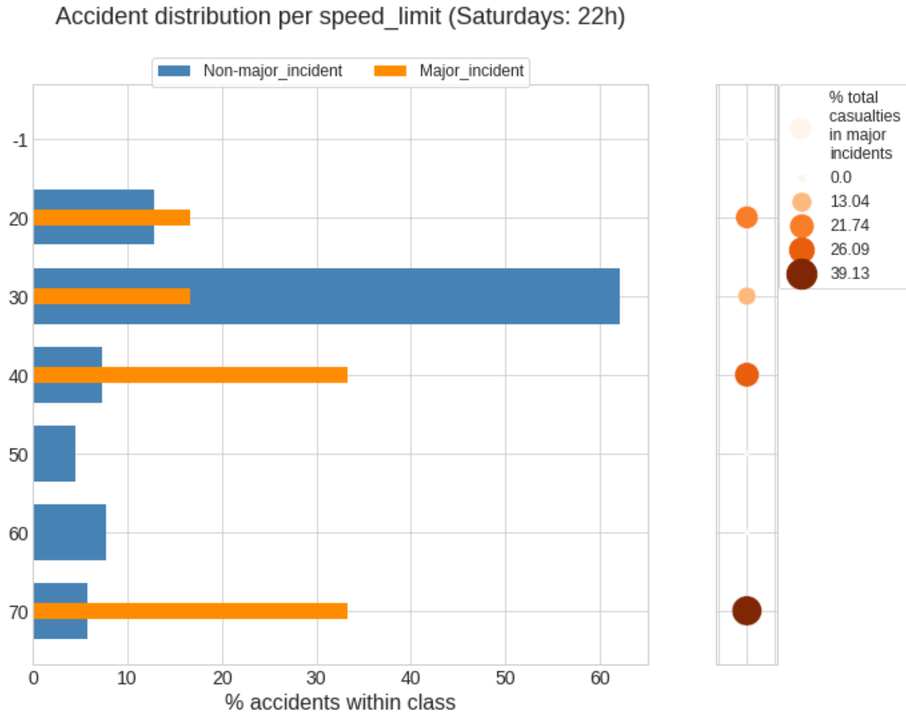
```
# Plot distribution of accidents and casualties per area on Saturdays at 22h
fig, ax = plt.subplots(1, 2, figsize=(10,8), sharey=True, \
                       gridspec_kw=dict(width_ratios=[3, 0.3]))

saturdays_22h = accident[(accident['day_of_week']==7)&(accident['hour']==22)]
plot_bullets('urban_or_rural_area', ax[0], ax[1], saturdays_22h, ' (Saturdays: 22h)')
```




```
# Plot distribution of accidents and casualties per
# first road class on Saturdays at 22h
fig, ax = plt.subplots(1, 2, figsize=(10,8), sharey=True, \
                       gridspec_kw=dict(width_ratios=[3, 0.3]))

saturdays_22h = accident[(accident['day_of_week']==7)&(accident['hour']==22)]
plot_bullets('first_road_class', ax[0], ax[1], saturdays_22h, ' (Saturdays: 22h)')
```

Accident distribution per first_road_class (Saturdays: 22h)

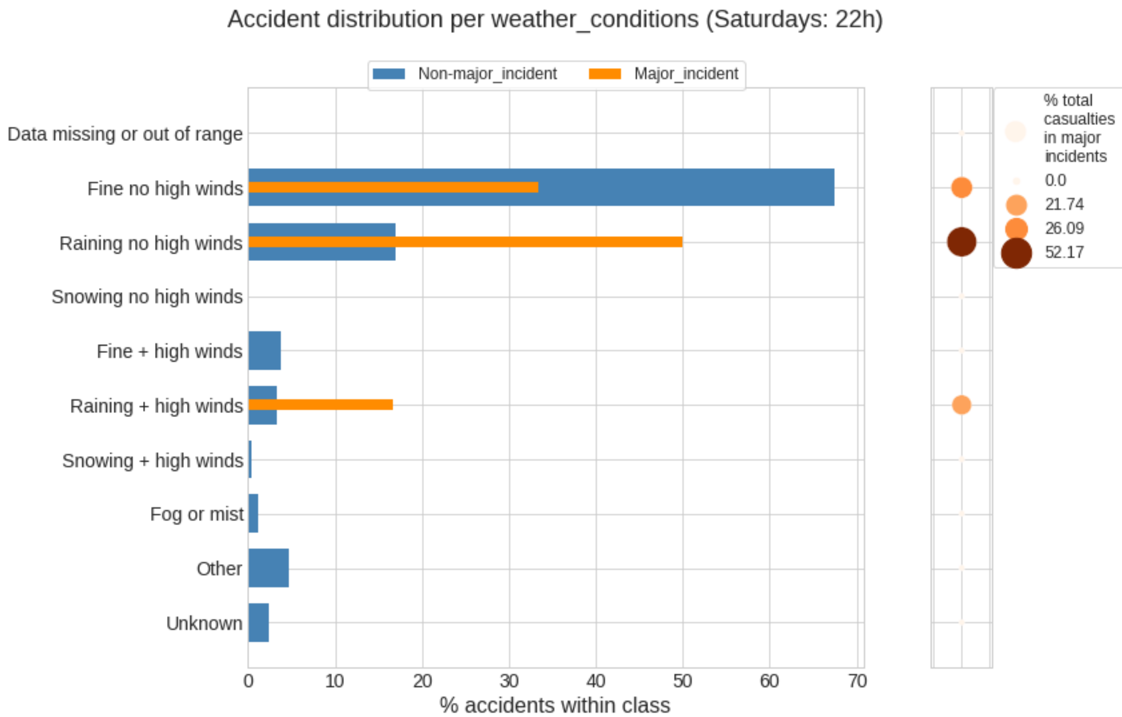
```
# Plot distribution of accidents and casualties
# per speed limit on Saturdays at 22h
fig, ax = plt.subplots(1, 2, figsize=(10,8), sharey=True, \
                       gridspec_kw=dict(width_ratios=[3, 0.3]))

saturdays_22h = accident[(accident['day_of_week']==7)&(accident['hour']==22)]
plot_bullets('speed_limit', ax[0], ax[1], saturdays_22h, ' (Saturdays: 22h)')
```




```
# Plot distribution of accidents and casualties
# per light condition on Saturdays at 22h
fig, ax = plt.subplots(1, 2, figsize=(10,8), sharey=True, \
                       gridspec_kw=dict(width_ratios=[3, 0.3]))

saturdays_22h = accident[(accident['day_of_week']==7)&(accident['hour']==22)]
plot_bullets('light_conditions', ax[0], ax[1], saturdays_22h, ' (Saturdays: 22h)')
```

Accident distribution per light_conditions (Saturdays: 22h)

```
# Plot distribution of accidents and casualties
# per waether conditions on Saturdays at 22h
fig, ax = plt.subplots(1, 2, figsize=(10,8), sharey=True, \
                       gridspec_kw=dict(width_ratios=[3, 0.3]))

saturdays_22h = accident[(accident['day_of_week']==7)&(accident['hour']==22)]
plot_bullets('weather_conditions', ax[0], ax[1], saturdays_22h, ' (Saturdays: 22h)')
```



Double-click (or enter) to edit

Predicting the next strike

Location and time are the most important predictors

```
# Use Decision Tree model to estimate feature importance
features = ['latitude', 'longitude', 'day_of_week', 'first_road_class', \
            'road_type', 'speed_limit', 'junction_detail', 'junction_control', \
            'second_road_class', 'pedestrian_crossing_human_control', \
            'pedestrian_crossing_physical_facilities', 'light_conditions', \
            'weather_conditions', 'road_surface_conditions', \
            'special_conditions_at_site', 'carriageway_hazards', \
            'urban_or_rural_area', 'month', 'hour']
# set the features
x = accident[features].copy()
# set the target
y = accident['major_incident']
# instantiate a decision tree classifier
tree = DecisionTreeClassifier(random_state = 0).fit(x, y)
# create a new dataframe with features and it importance
important_features = pd.DataFrame({'features':x.columns,\
                                   'importance':tree.feature_importances_})


# check the most important features
important_features.sort_values('importance', ascending = False).head()
```

	features	importance
1	longitude	0.355722
0	latitude	0.310133
18	hour	0.072538

```
# check the least important features
important_features.sort_values('importance', ascending = True).head()
```

	features	importance
14	special_conditions_at_site	0.000046
9	pedestrian_crossing_human_control	0.000940
10	pedestrian_crossing_physical_facilities	0.006904
5	speed_limit	0.009346
4	road_type	0.009481

```
cas_0.head()
```

```
0    0.000009
1    0.000017
2    0.000009
3    0.000009
4    0.000017
Name: number_of_casualties, dtype: float64
```