2022

Big Data and Data Mining Project

Code: 771762

ABIODUN OWOEYE

UNIVERSITY OF HULL | Artificial Intelligence & Data Science

Contents

Introduction

"The United Kingdom's Department of Transportation yearly releases statistics and information about reported accidents and casualties on public roads" (GOV.UK, 2022). This report is based on Great Britain's 2019 accident dataset.

The dataset consists of three categories.

1. Accident
2. Casualties
3. Vehicle

Accident severity is documented in three classes:

1. Fatal – 1
2. Serious – 2
3. Slight - 3

We will analyze and evaluate varying conditions contributing to accidents - where, how, etc. analyzing the road exigencies for recommendation such as the conditions of the roads during accidents, investigating the area and how activities (e.g. sporting activities) within the communities contribute to the need for specific actions. Principal component analysis (PCA) was used to determine the best attributes of our data to be modelled, we will then progress to build machine learning ensembled models, stacked to predict these features against our label, accident severity. Lastly, we will compare our results with the government's published report.

Analysis

Data Preprocessing and Cleaning

The accidents, casualties, and vehicles dataset require a significant amount of cleaning to prepare data for analysis. Kindly find below the process of cleaning.

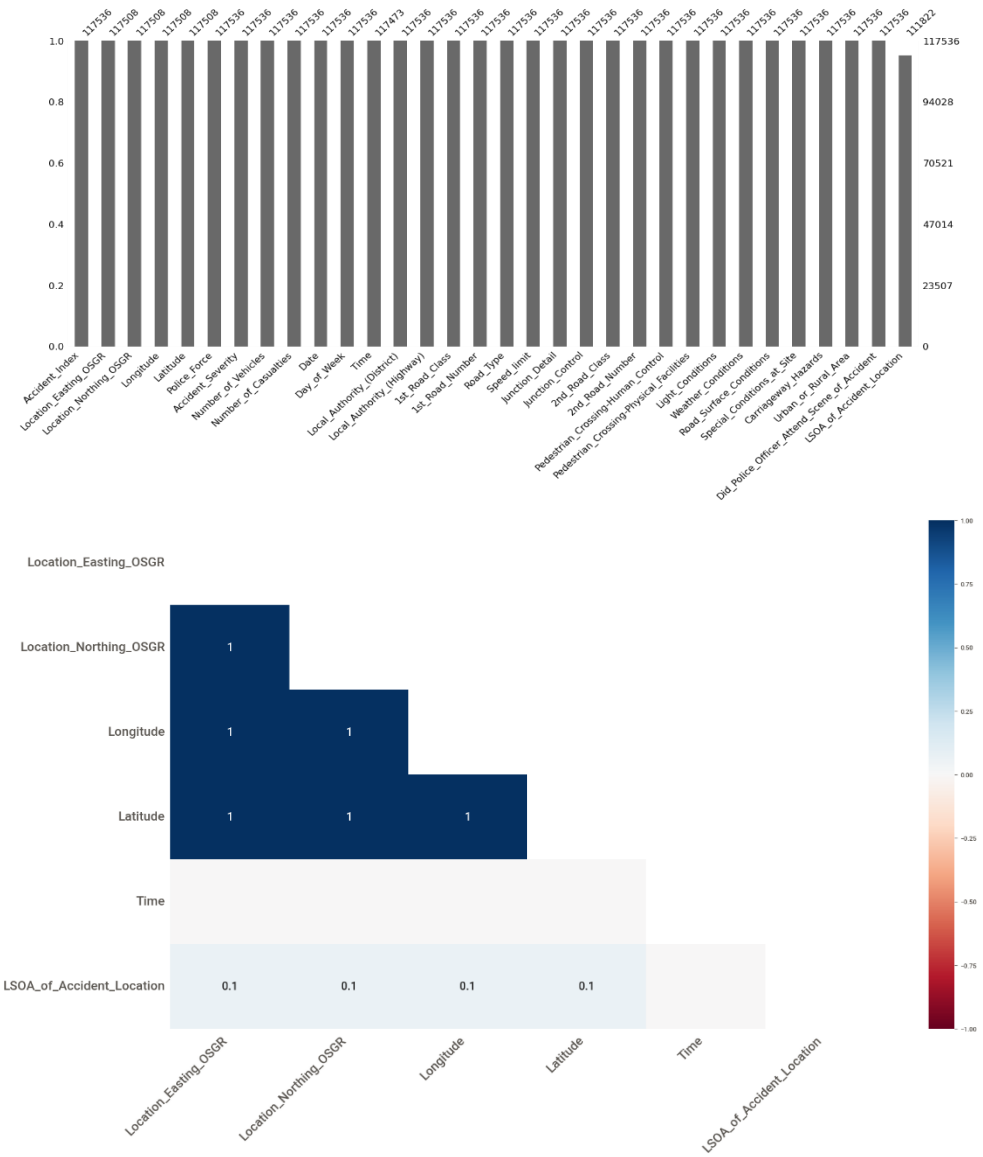


Figure 1: Correlation (Missing Data)

There are five links in accident data with missing values. The police force correlated with these attributes because the values are ***location based*** as others and it is the key to cleaning these other columns. To clean the columns:

- we proceeded to getting the values of the latitude and longitude by using the geopy library; a google maps library in python to locate the coordinates of addresses (GeoPy, 2018). The column Police Force location addresses are stored in our variable

lookup, we used geopy to generate the latitude and longitude by searching a 3mile radius.

- In easting and northing, we filled the nan with a random number (737), used the longitude and latitude produced to generate the easting and northing values on those corresponding lines using the Bidirectional UTM-WGS84 converter for python (Pypi, 2020).
- To clean the LSOA_of_Accident_Location, we used the publicly available data sourced from United Kingdom's Ministry of Housing and Communities & Local Government to populated the 28 missing rows correlated with the other columns (Ministry of Housing, Communities & Local Government, 2018).
- We created a python function to populate these cells in our dataset.

```python
from geopy.geocoders import Nominatim
from geopy.distance import distance

from geopy.geocoders import Nominatim


def find_location(spot, radius):
    geolocator = Nominatim(user_agent="GoogleV3")
    location = geolocator.geocode(spot)
    print(location.address)

    print((location.latitude, location.longitude))
    print(location.raw)

    boundary_cordinates = []  # List to store the cordinates

    angles = (0, 90, 180, 270)
    for line in angles:
        d = distance(kilometers=radius).destination(
            (location.latitude, location.longitude), bearing=line
        )
        boundary_cordinates.append((d.latitude, d.longitude))
    return boundary_cordinates

# Search radius
rad = 8
```

```python
dist = find_location("Lancashire \n\n", rad)
print("Coordinates for Lancashire", dist)
print("\n")
dist = find_location("Merseyside \n\n", rad)
print("Coordinates for Merseyside", dist)
print("\n")
dist = find_location("Cheshire, United Kingdom \n\n", rad)
print("Coordinates for Cheshire", dist)
print("\n")
dist = find_location("North Yorkshire \n\n", rad)
print("Coordinates for North Yorkshire", dist)
print("\n")
dist = find_location("West Yorkshire \n\n", rad)
print("Coordinates for West Yorkshire", dist)
print("\n")
dist = find_location("Humberside, England \n\n", rad)
print("Coordinates for Humberside", dist)
print("\n")
dist = find_location("Warwickshire \n\n", rad)
print("Coordinates for Warwickshire", dist)
print("\n")
dist = find_location("Kent \n\n", rad)
print("Coordinates for Kent", dist)
print("\n")
dist = find_location("Sussex \n\n", rad)
print("Coordinates for Sussex", dist)
```

Result:

```
Lancashire, England, United Kingdom
(53.8611703, -2.5650887919475496)
{'place_id': 329892012, 'licence': 'Data © OpenStreetMap contributors, ODbL 1.0. https://osm.org/copyright', 'osm_type': 'relat
ion', 'osm_id': 118082, 'boundingbox': ['53.4827602', '54.2395575', '-3.0845488', '-2.0450727'], 'lat': '53.8611703', 'lon': '-
2.5650887919475496', 'display_name': 'Lancashire, England, United Kingdom', 'class': 'boundary', 'type': 'ceremonial', 'importa
nce': 0.7526028209470421}
Coordinates for Lancashire [(53.93304611574668, -2.5650887919475496), (53.86110870736586, -2.443496846948984), (53.789293618020
935, -2.5650887919475496), (53.86110870736586, -2.686680736946115)]
```

```python
import utm  # Library to convert longitude and latitude to easting and northing


def nan_replacement(LSOA, Latitude, Longitude, police_code):
    accidents.loc[
        (accidents["Latitude"].isna()) & (accidents["Police_Force"] == police_code),
        "Latitude",
    ] = Latitude
    accidents.loc[
        (accidents["Longitude"].isna()) & (accidents["Police_Force"] == police_code),
        "Longitude",
    ] = Longitude
    accidents.loc[
        (accidents["LSOA_of_Accident_Location"].isna())
        & (accidents["Police_Force"] == police_code),
        "LSOA_of_Accident_Location",
    ] = LSOA

    easting, northing, zone, ut = utm.from_latlon(
        Latitude, Longitude
    )  # Converting lat and lon to XY cordinates

    accidents["Location_Easting_OSGR"].replace(
        737, easting, inplace=True
    )  # replaces 737 with the value of easting
    accidents["Location_Northing_OSGR"].replace(
        737, northing, inplace=True
    )  # replaces 737 with the value of northing
```

```python
print(
    "Lancashire coordinate: ",
    nan_replacement("E04005283", 53.8611703, -2.5650887919475496, 4),
)
print(
    "Merseyside coordinate: ",
    nan_replacement("E08000012", 53.4953602, -2.973937960086006, 5),
)
print(
    "Cheshire coordinate: ",
    nan_replacement("E01032930", 53.235652099999996, -2.489152998049048, 7),
)
print(
    "North Yorshire coordinate: ",
    nan_replacement("E01033183", 54.13453275, -1.498628491239545, 12),
)
print(
    "West Yorshire coordinate: ",
    nan_replacement("E05001393", 53.741427099999996, -1.7202003766777243, 13),
)
print(
    "Humberside coordinate: ",
    nan_replacement("E01032595", 53.5713963, -0.35346073234607067, 16),
)
print(
    "Warwickshire coordinate: ",
    nan_replacement("E01031267", 52.32130635, -1.5536905536661392, 23),
```

Result:

```
Lancashire coordinate:  ('E04005283', 528602.9177658462, 5968163.253291729, 53.8611703, -2.5650887919475496)
Merseyside coordinate:  ('E08000012', 501728.94924505206, 5927377.887761338, 53.4953602, -2.973937960086006)
Cheshire coordinate:  ('E01032930', 534096.0499697446, 5898607.172382786, 53.235652099999996, -2.489152998049048)
North Yorshire coordinate:  ('E01033183', 598093.1154113626, 5999531.637885479, 54.13453275, -1.498628491239545)
West Yorshire coordinate:  ('E05001393', 584407.1896826852, 5955513.538098047, 53.741427099999996, -1.7202003766777243)
Humberside coordinate:  ('E01032595', 675238.1903010518, 5939094.078500858, 53.5713963, -0.35346073234607067)
Warwickshire coordinate:  ('E01031267', 598574.8608830112, 5797760.579239913, 52.32130635, -1.5536905536661392)
Kent coordinate:  ('E01024076', 340807.87805185607, 5675320.990790076, 51.20707485, 0.7210361813401444)
Sussex coordinate:  ('E01031802', 706366.717842268, 5647764.559529817, 50.94451385000001, -0.062267328417847545)
Dorset coordinate:  ('E01020373', 546180.1805140063, 5627437.4097990915, 50.79683685, -2.34473226124306)
South Wales coordinate:  ('E02003805', 510048.7020711491, 5892527.817758741, 53.1820038, -2.8496329)
North Wales coordinate:  ('Unknown', 477382.29381035245, 5721548.256652777, 51.64448175, -3.326878750000005)
Dyfed Powys coordinate:  ('Unknown', 472176.3114779589, 5777670.039099048, 52.1488332, -3.4066453)
```

- Replaced the leftover missing LSOA data with the mode of the column.
- Replaced nan values with the mode in the Time column

Replacing the -1, the exported value for NULL or out of range values:

The series with -1 was put into a list and created a function to effectively replace them with the mode, average, 0 depending on the specifics of the column. There were columns whose mode was already -1, they were filtered for the mode of the rest of the columns that are not -1 to replace the -1.

After cleaning, datasets were merged into one.

```python
l = [
    accidents["Junction_Control"],
    accidents["2nd_Road_Class"],
    accidents["2nd_Road_Number"],
]


def out_of_range(series, l):
    for element in series:
        for i in range(len(element)):
            if element[i] == -1:
                i = element.replace([-1], element.mode(), inplace=True)

        print(element)

    for elem in l:
        for x in range(len(elem)):
            if elem[x] == -1:
                x = elem.replace([-1], 0, inplace=True)
        print(elem)


out_of_range(series, l)
```

```python
c = [
    casualties["Sex_of_Casualty"],
    casualties["Age_of_Casualty"],
    casualties["Age_Band_of_Casualty"],
    casualties["Pedestrian_Location"],
    casualties["Car_Passenger"],
    casualties["Bus_or_Coach_Passenger"],
    casualties["Pedestrian_Road_Maintenance_Worker"],
    casualties["Casualty_Type"],
    casualties["Casualty_Home_Area_Type"],
    casualties["Casualty_IMD_Decile"],
]

# could not apply the above function because it takes two arguments whil
def my_function(c):
    for element in c:
        for i in range(len(element)):
            if element[i] == -1:
                i = element.replace([-1], element.mode(), inplace=True)

        print(element)


my_function(c)
```

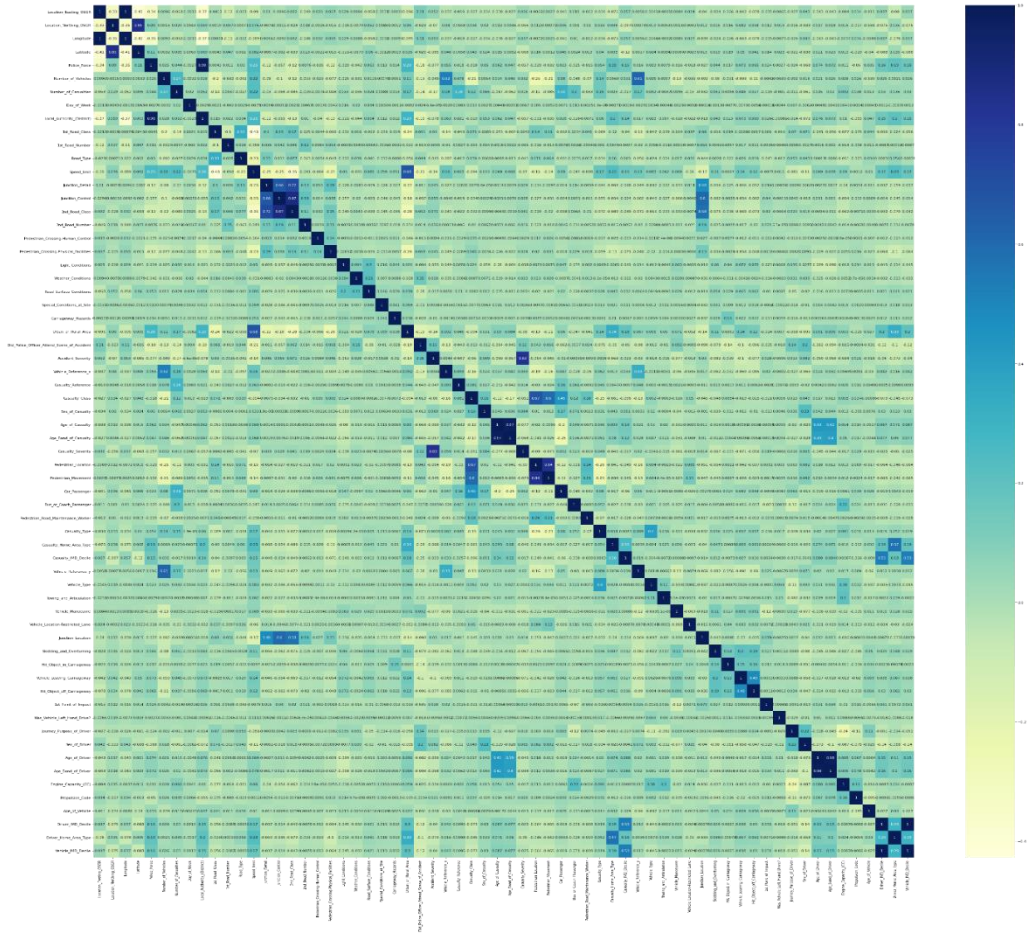
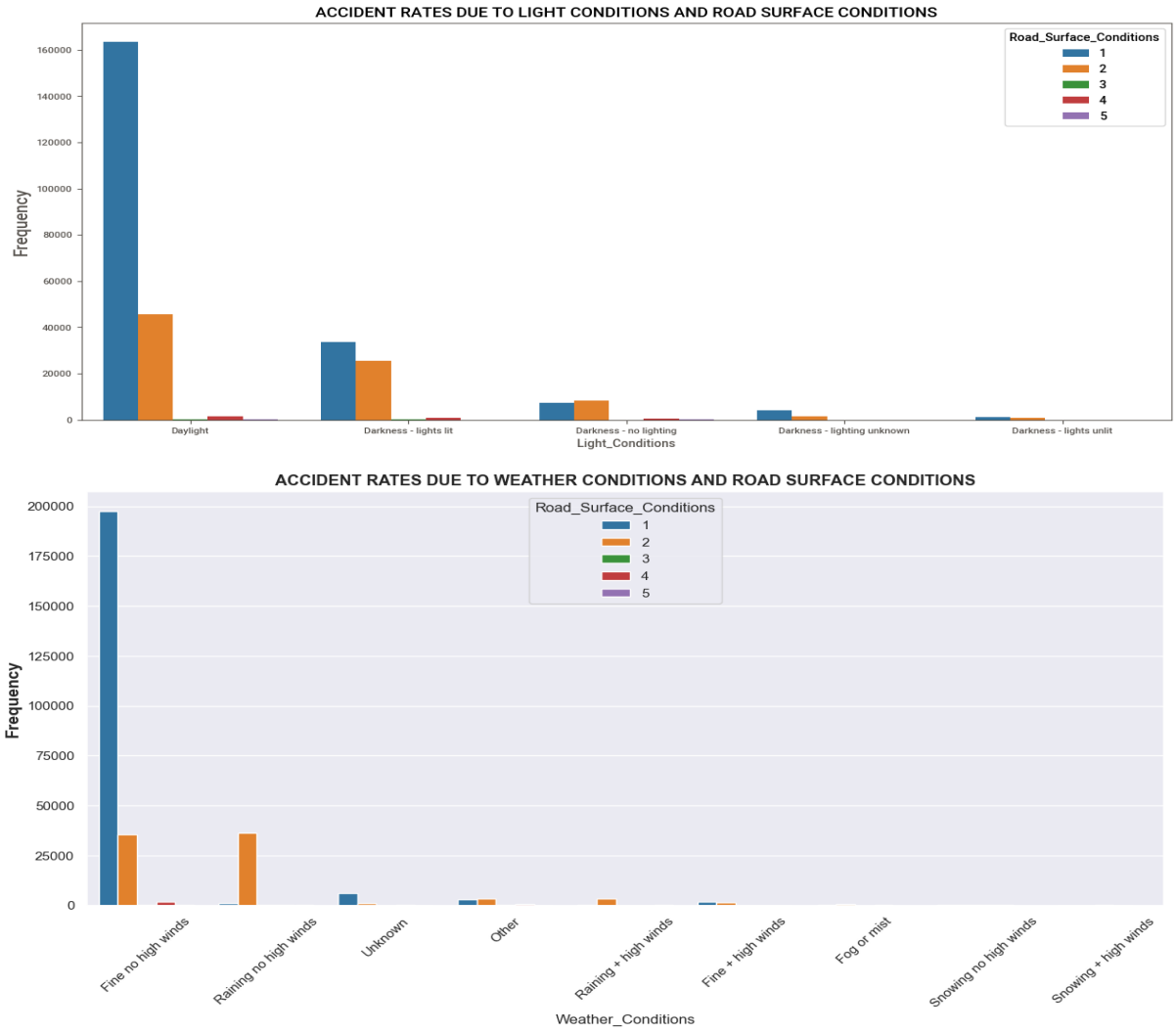


Figure 2: Relationships in dataset

Figure 2 revealed some relationships that we will explore below:

- Light conditions-weather conditions-road surface conditions,
- Speed limit-junction detail-junction control-2nd Road class – Pedestrian human crossing-Pedestrian crossing physical facilities
- Skidding and overturning – Hit object in carriageway – vehicle leaving carriageway – Hit object off the carriageway
- Driver IMD Decile - Driver Home Area – Vehicle IMD Decile
- Journey purpose of driver – Sex of Driver.

The plot below reveals 25000- 40,000 accidents occurred on wet ground on a rainy day.



. Figure 3: Light, Weather Conditions & Road Surface

Road Type and Accident Severity: The plot reveal that most accident occurred on single carriageway (Mostly farm roads without foot or bikers' pathways).

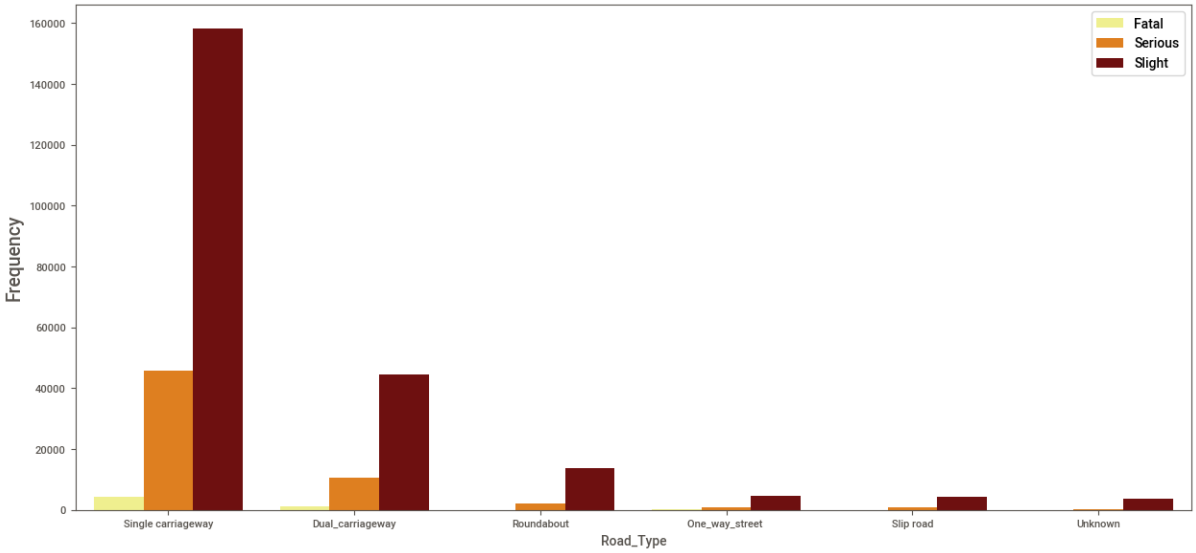


Figure 4: Road Type & Severity

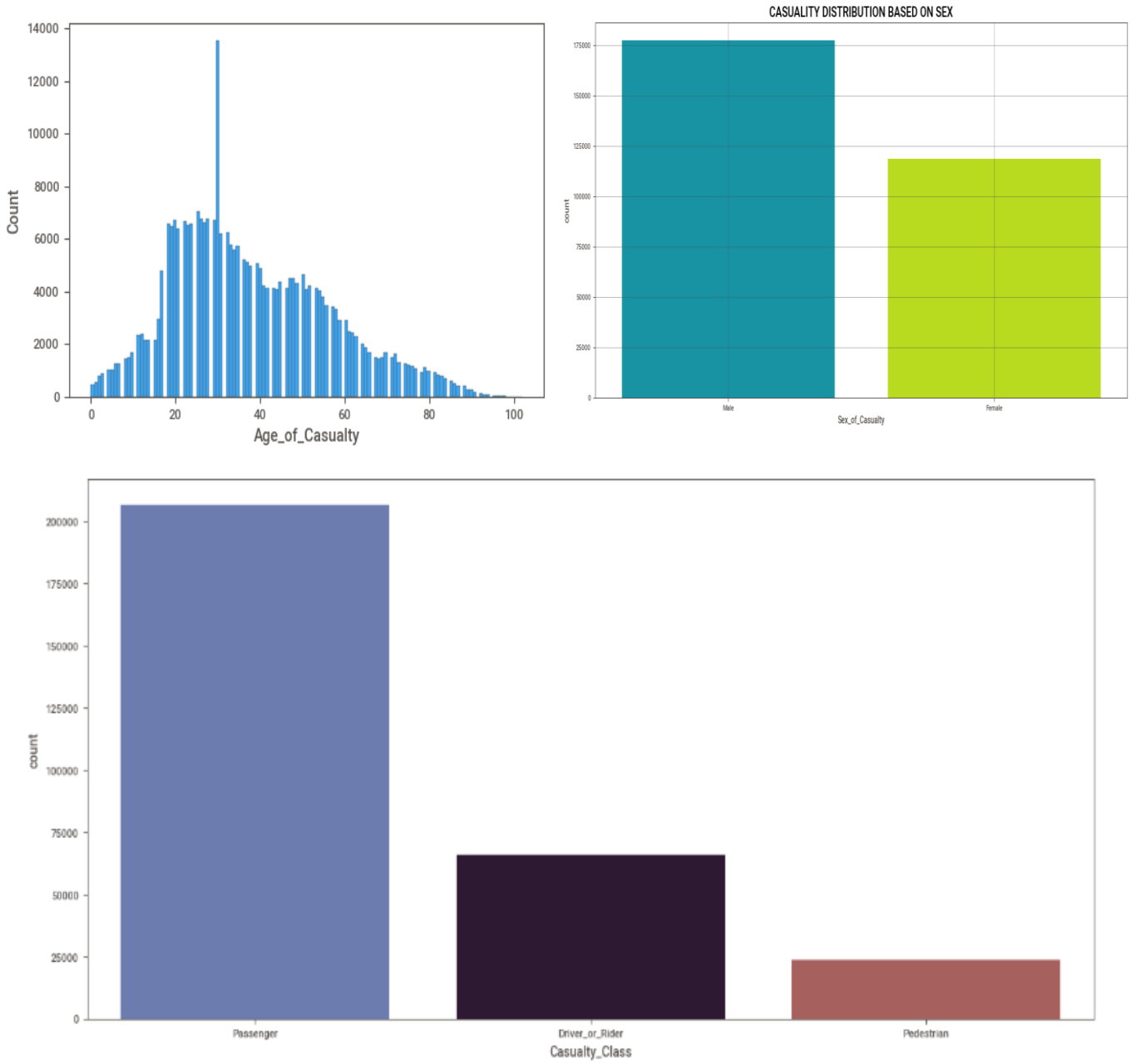


Figure 5: Casualties Distribution

Most casualties are Passengers, Male and between 26-35 years old.

Urban or Rural areas – Speed Limit – Casualty Severity:

Speed limit during most accidents was 30, severity slight in rural or urban areas and largely on dry road surface conditions

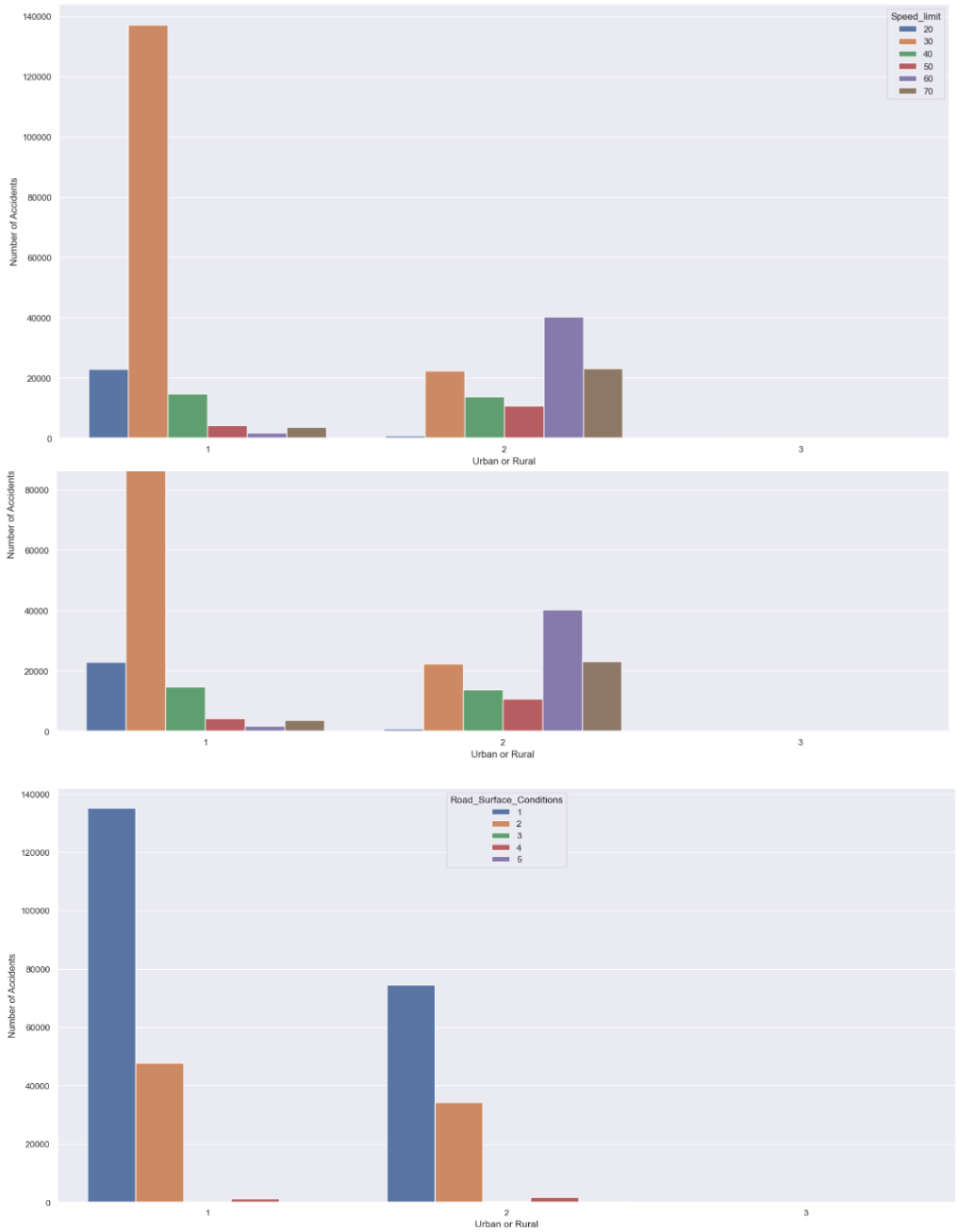


Figure 6: Urban or Rural Area

Accident Severity

The distribution of our accident severity is 77% Non-injurious, 20% Serious and 1.9% Fatal.

```
dataset["Accident_Severity"].value_counts() / len(dataset) * 100
```

```
3    77.595499
2    20.438529
1     1.965972
Name: Accident_Severity, dtype: float64
```

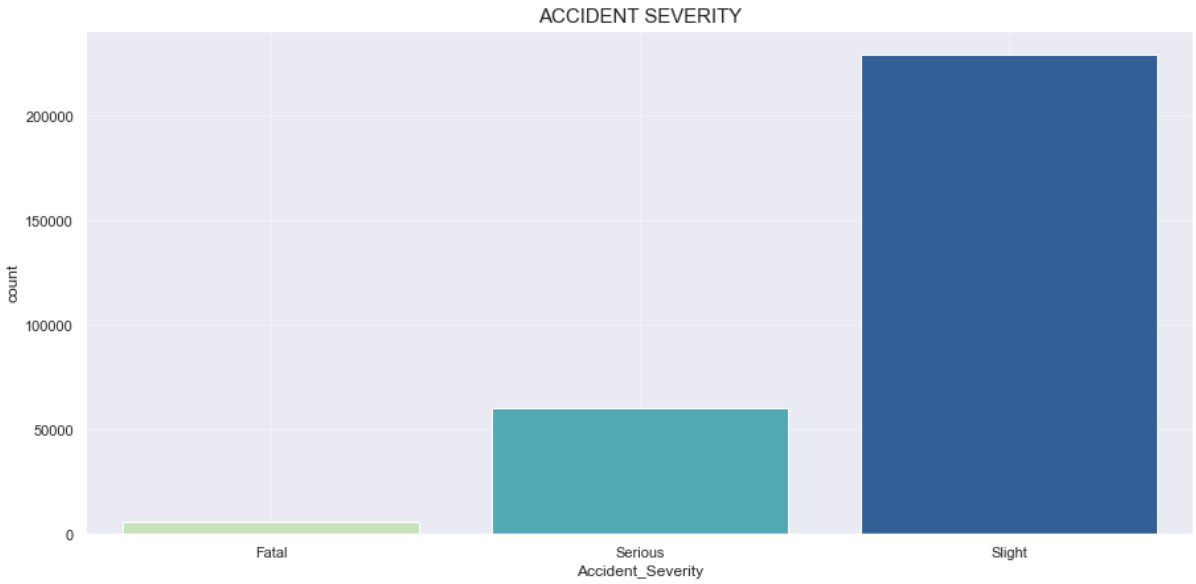


Figure 7: Accident Severity

Analysis

(a) Are there significant hours of the day, and days of the week, on which accidents occur?

Figure (8) reveals that majority of accidents occurred between the hours 6am-10am morning rush hour and 15:00pm -19:45pm.

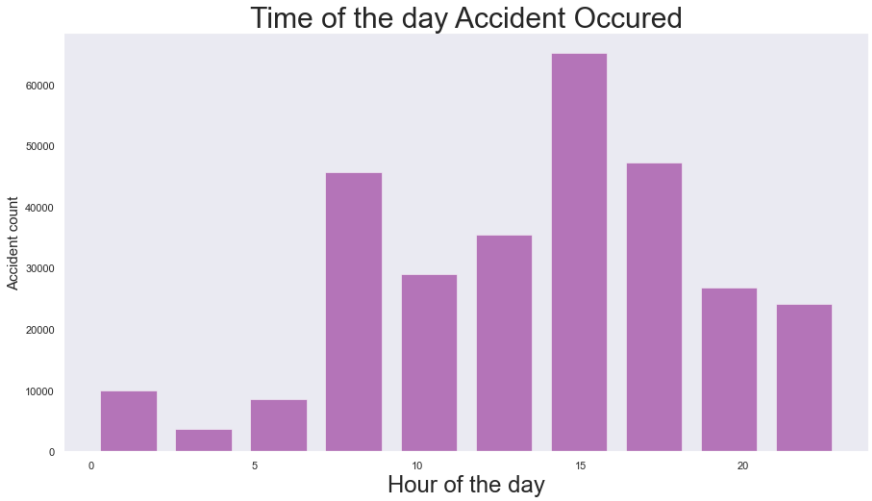


Figure 8: Time of Accidents

Figure (9) shows that accident occurrence is nearly evenly distributed across all days of the week, with Fridays with the most frequent accidents.

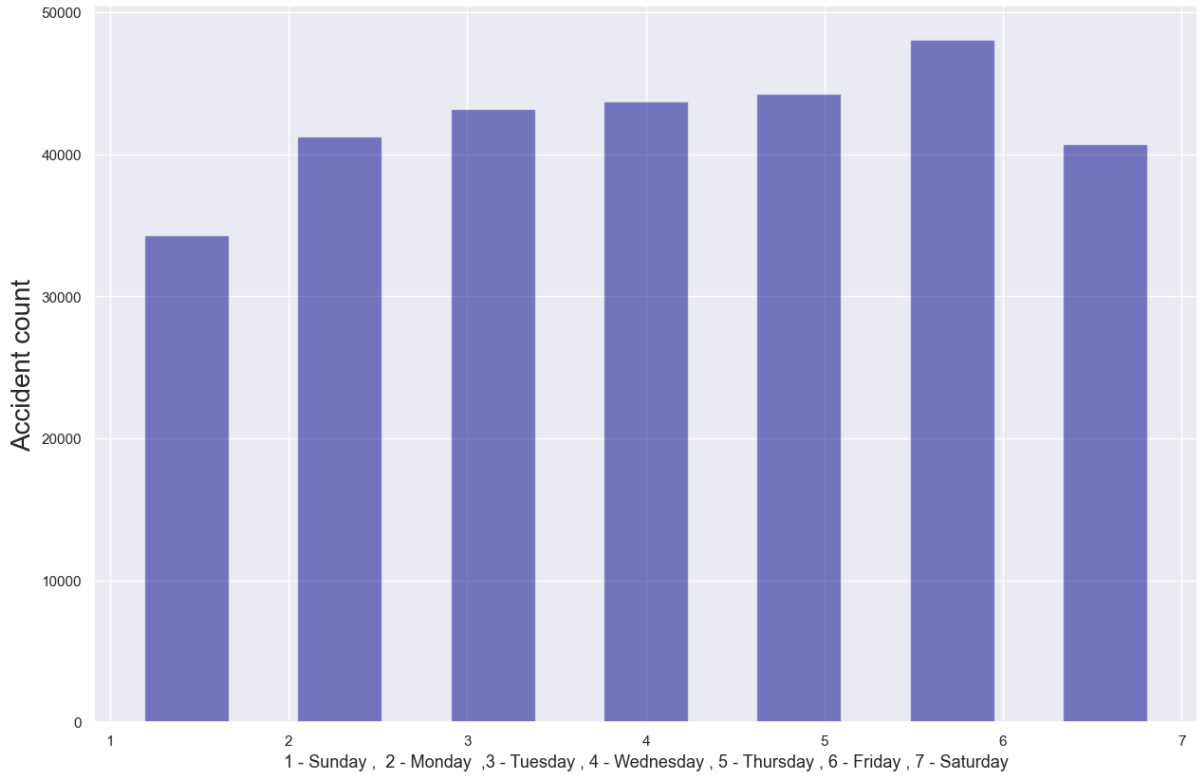


Figure 9: Accidents (Days of the Week)

(b) For motorbikes, are there significant hours of the day, and days of the week, on which accidents occur?

Pedal cycle appears to have the most motorbike accidents along The Broyle, Turnpike Farm, Ringmer, Lewes, East Sussex, South East England, United Kingdom.

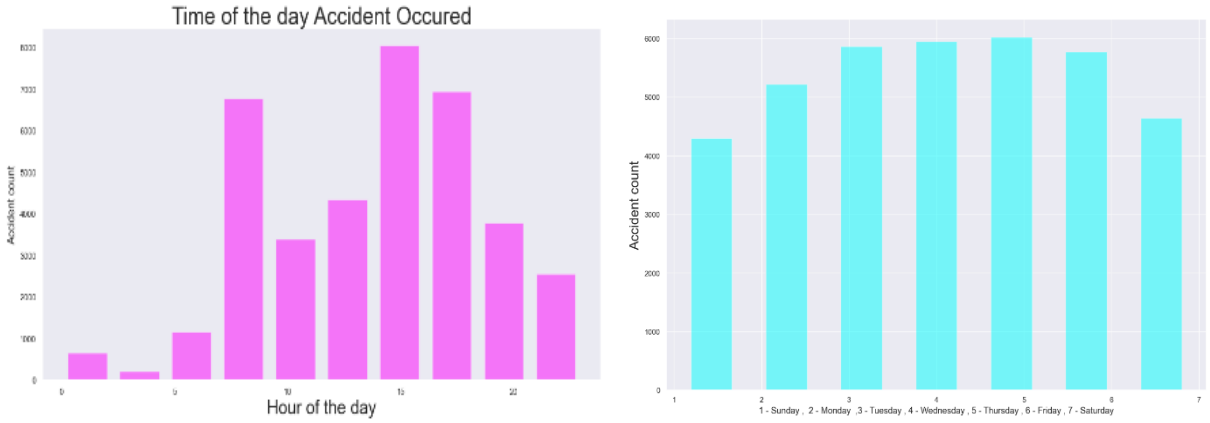


Figure 10: Time, Days of Accidents (Motorbikes)

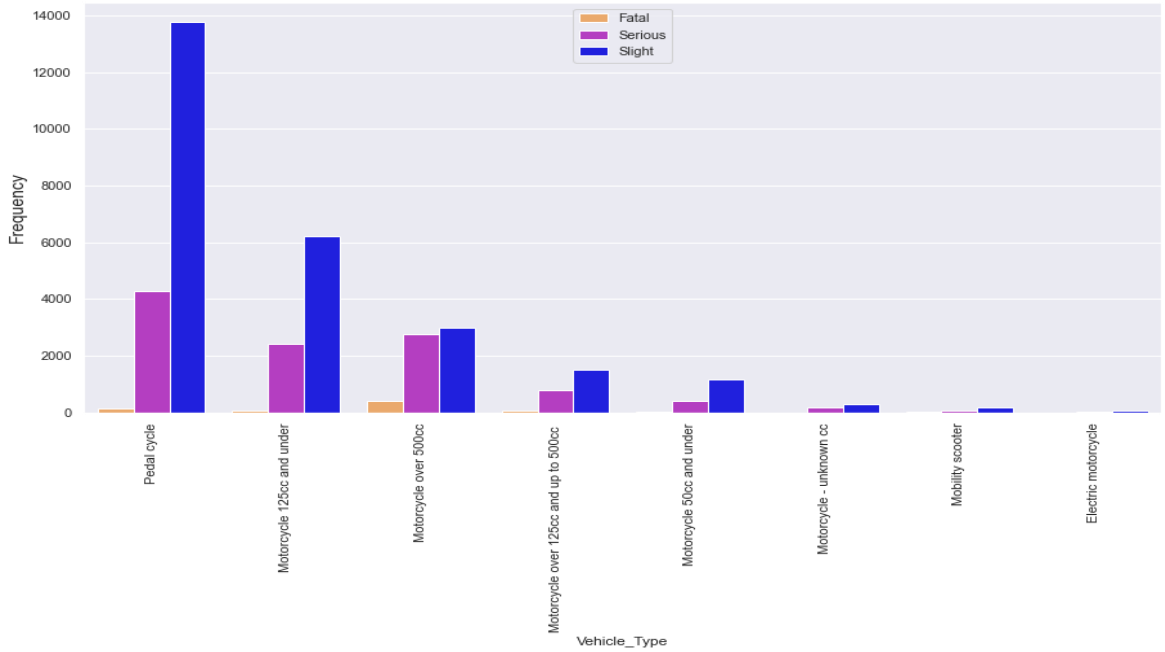


Figure 11: Accident Severity (Motorbikes)

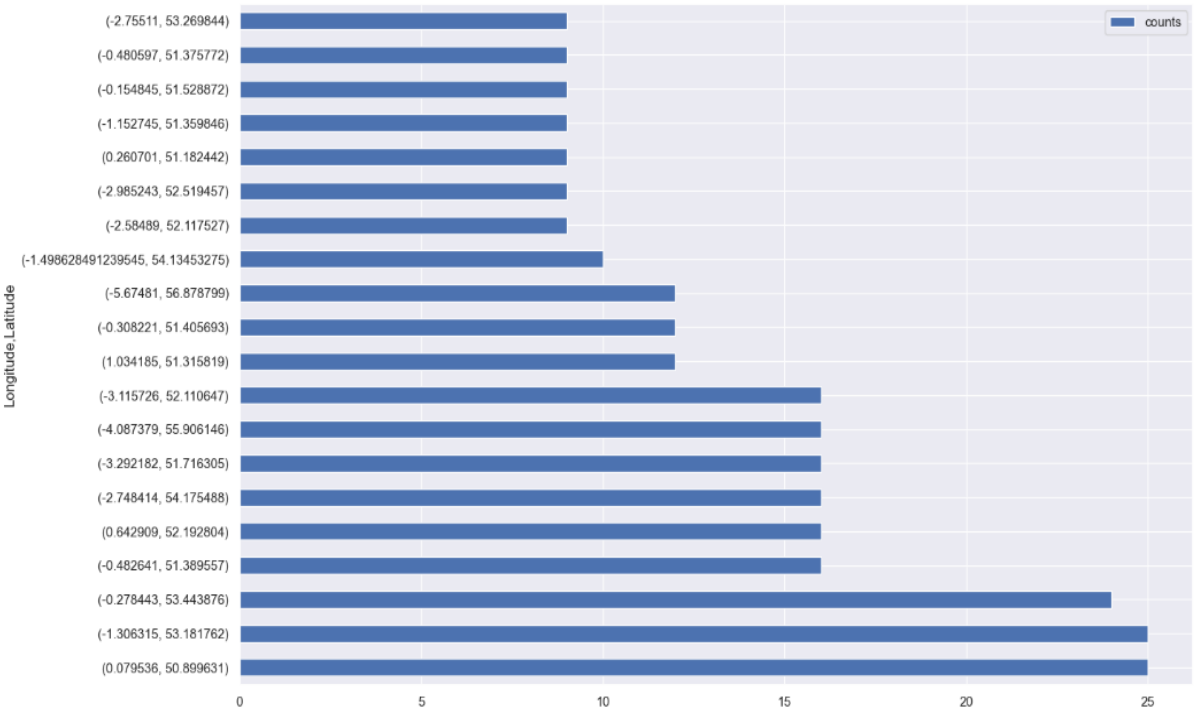


Figure 12: Accident Location (Motorbikes)

179 accidents were recorded on a single day on 19/09/2019. These are mostly farm roads built as Single carriageway (6).

(c) For pedestrians involved in accidents, are there significant hours of the day, and days of the week, on which they are more likely to be involved?

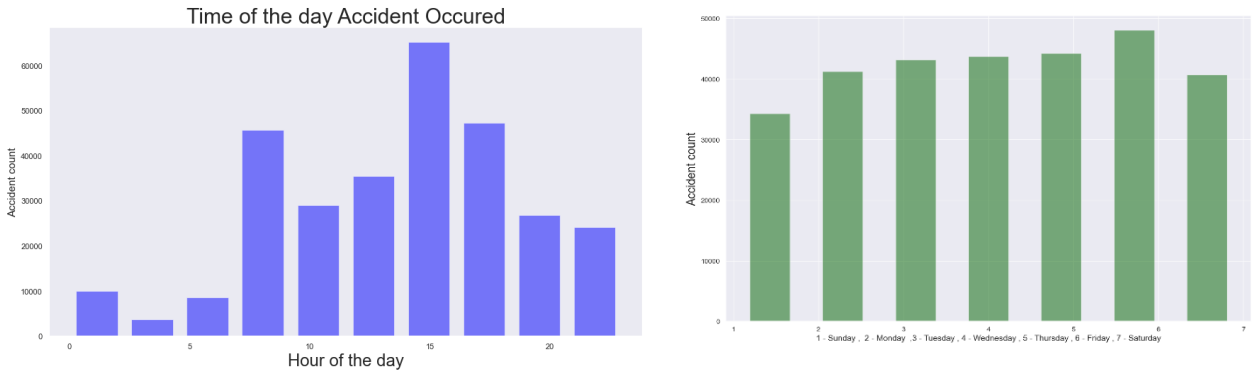


Figure 13: Time, Days of Accidents (Pedestrians)

Pedestrians accidents are more likely to happen on Fridays, from 3pm to 5pm. Most pedestrian accidents (88) happened along M60, Daisy Nook, Oldham, Woodhouses, Greater Manchester, a walking distance to Etihad Stadium on matchdays. This could have occurred due to stampede from people leaving the stadium.

(d) What impact, if any, does daylight savings have on road traffic accidents in the week after it starts and stops?

In 2019, daylight saving in the United Kingdom started 31/03/2019 01:00:00 and ended 27/Oct/2019 02:00:00. The number of accidents in the UK a week after daylight savings started was 5148 and 5518 after it ended.

There was 7.2% increase in accident rate during longer nights; over a 100% increase in fatal accident between that time (53-112).

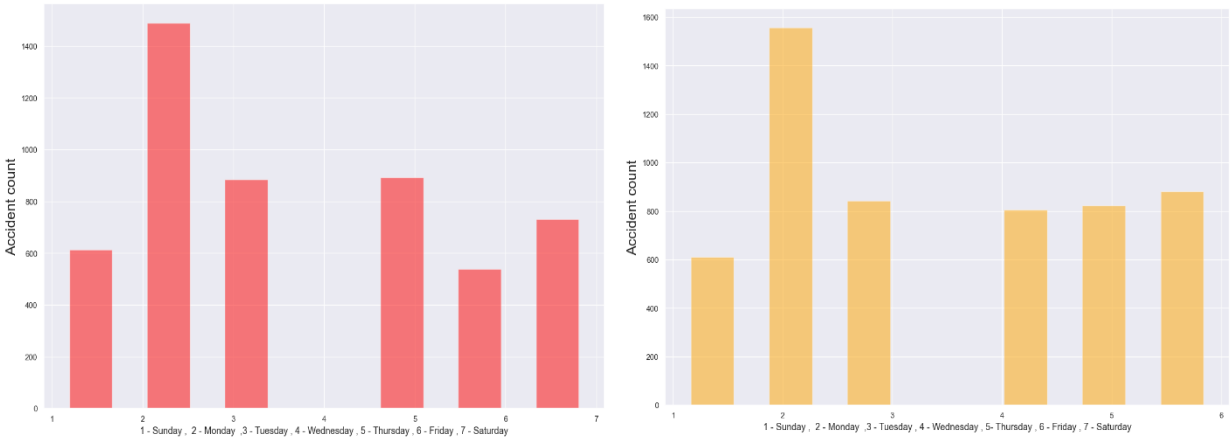


Figure 14: Daylight Savings (Before & After)

(e) What impact, if any, does sunrise and sunset times have on road traffic accidents?
Time of sunrise and sunset varies all year long. We focused on the daylight savings period which was 211/365 days. The sun rises between 5am-6am and sets between 19:53:00 and 20:53:00. The number of accidents recorded within the hour the sun rises is 3376 and during sunset is 5275. The plot below shows the week daylight savings started (L) and the week after it ended(R).

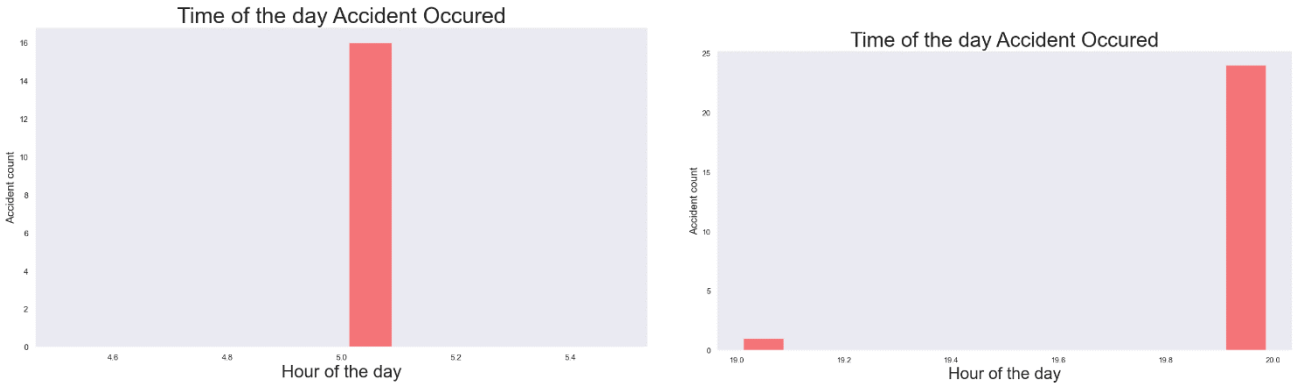


Figure 15: Sunrise & Sunset Accidents

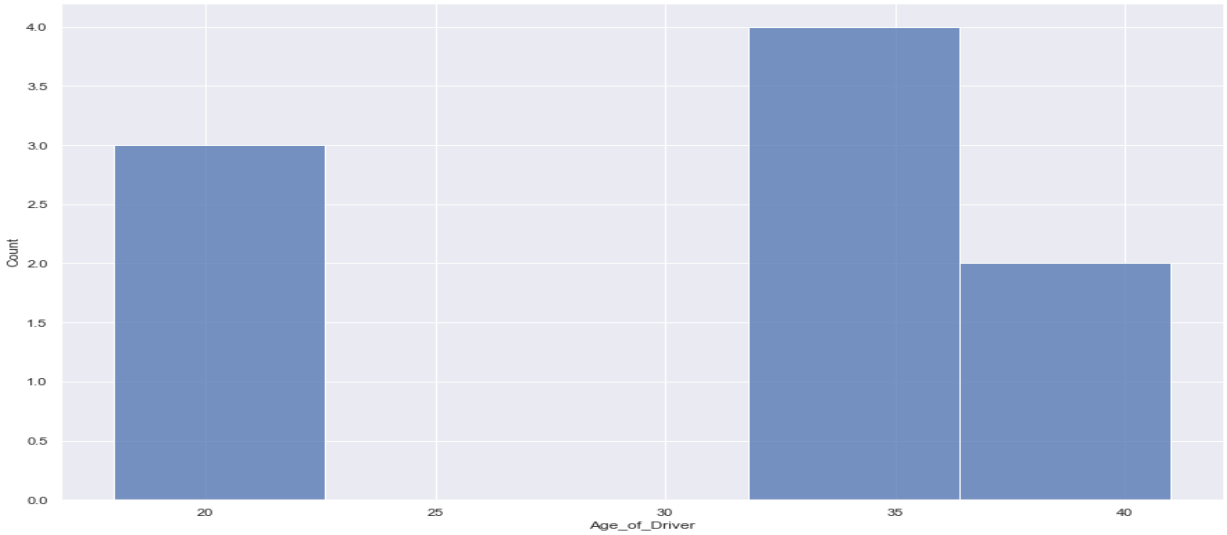


Figure 16: Sunrise (Age of Driver)

Middle aged population were more involved in accidents, weather was fine during these times, accidents to seniors rose steeply up at sunset.

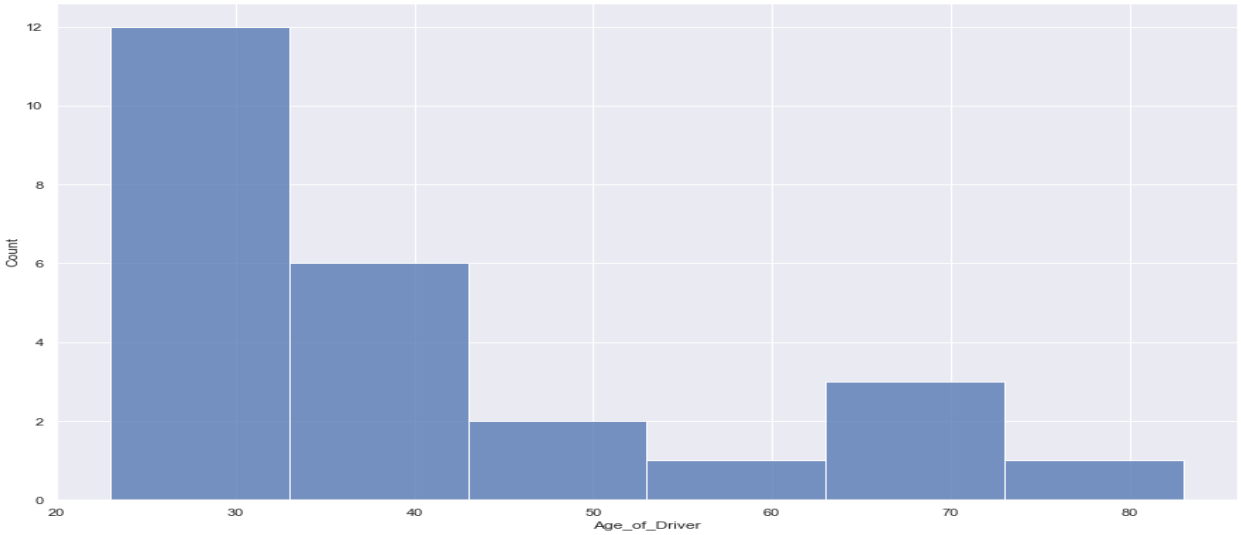


Figure 17: Sunset (Age of Driver)

(f) Are there particular types of vehicles that are more frequently involved in road traffic accidents?

The vehicle type with frequent accidents are Cars, age 6 years, engine capacity 1598, accident severity, slight.

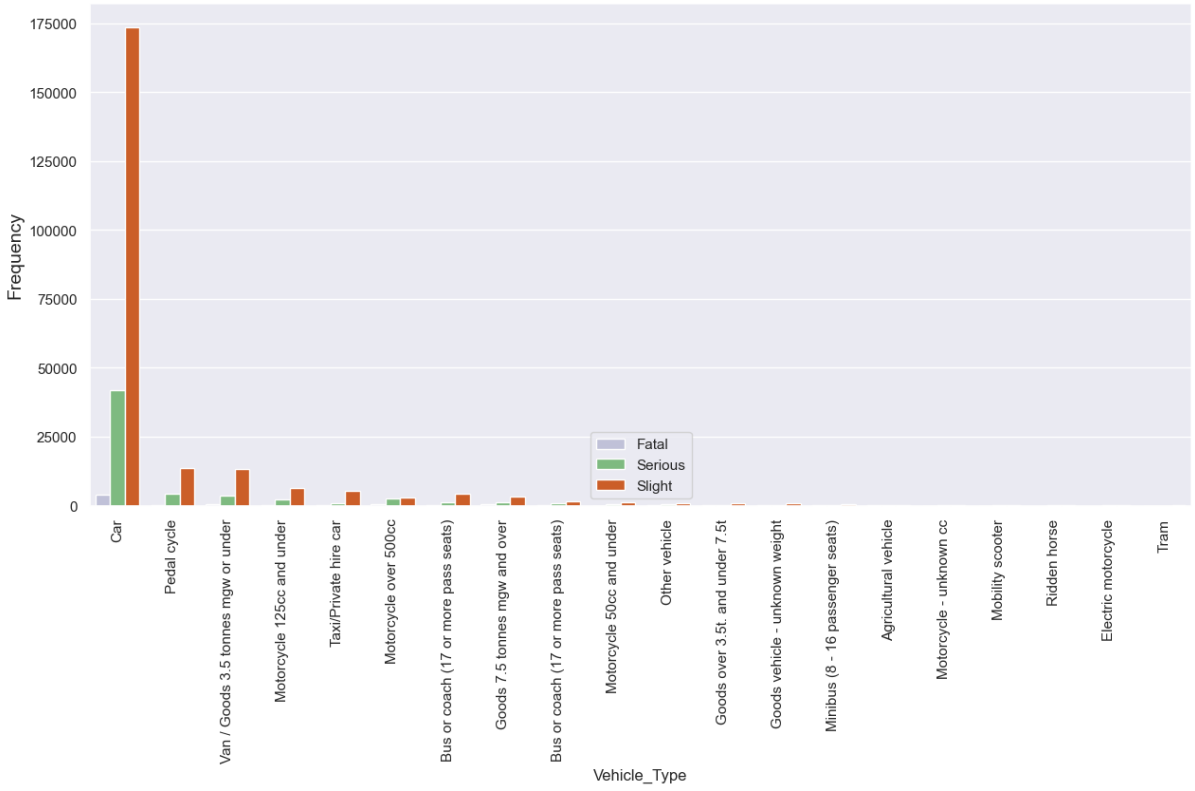


Figure 18: Vehicle Related Accidents

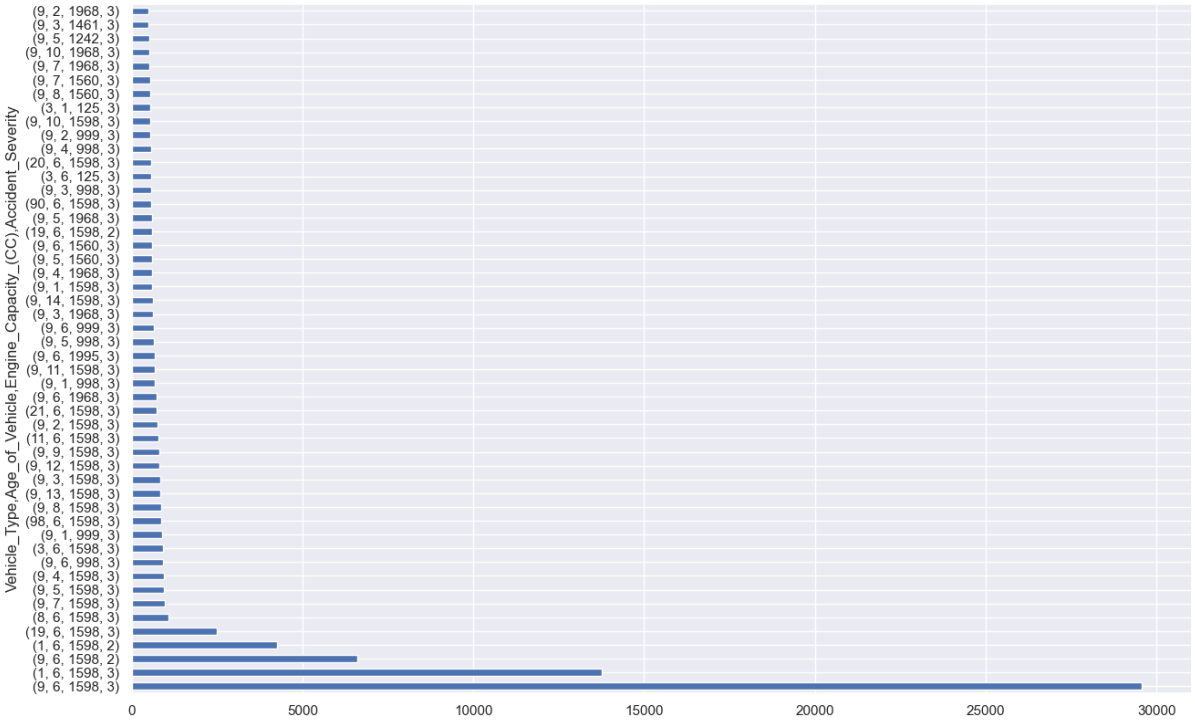


Figure 19: Vehicle Related Variable

(g) Are there particular conditions (weather, geographic location, situations) that generate more road traffic accidents?

Top 50 geographic location with the highest accidents count.

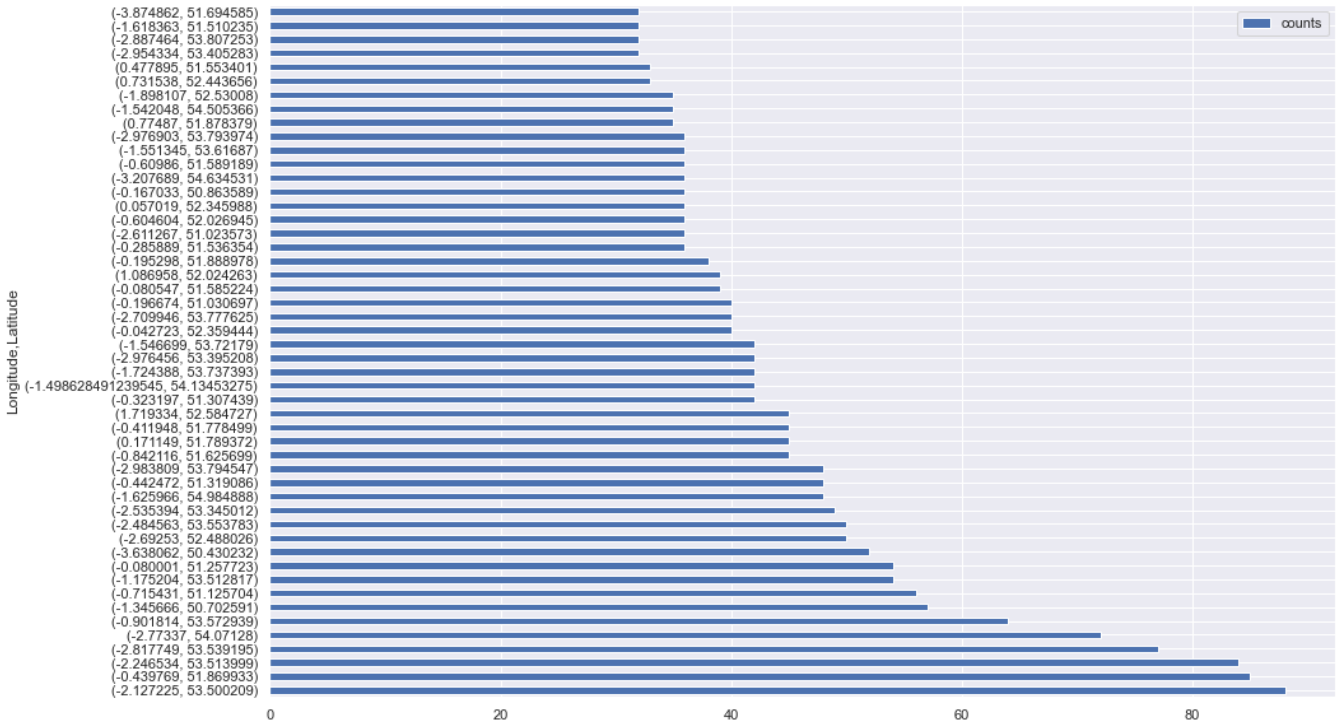


Figure 20: Top 50 Accident Locations

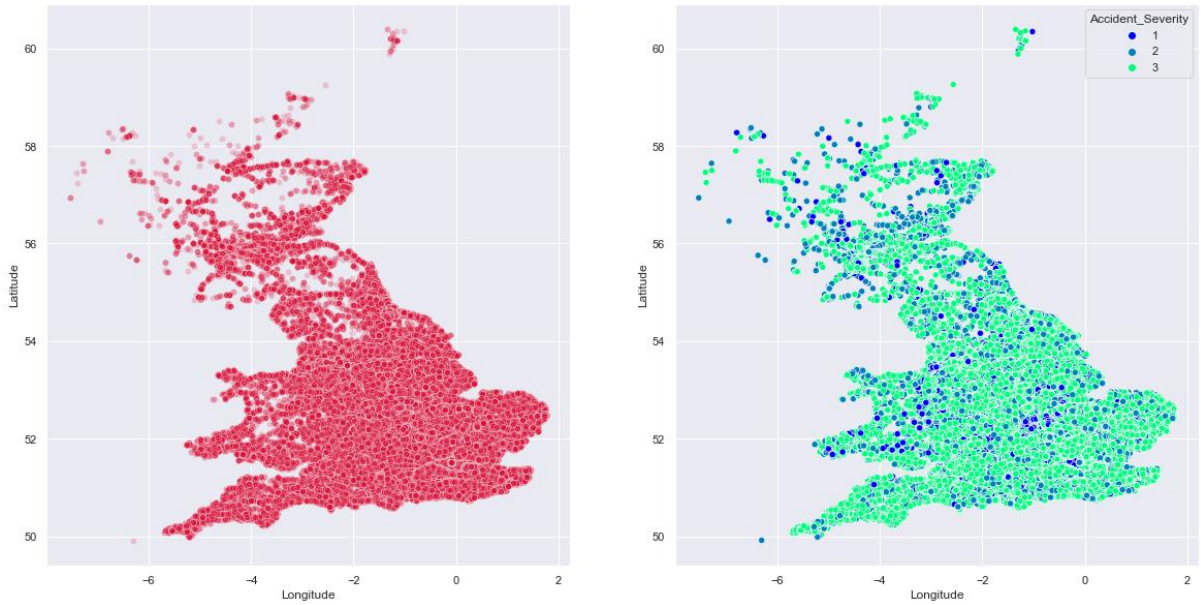


Figure 21: Accident Locations

Figure 22: Casualties Hotspot

The plot above shows that London, Bristol, Manchester, Glasgow and some few other areas are the hotspot of most accidents.

Maps of Accident Hotspots

Using google API we generated maps of the accident hotspots in the UK.

```
import gmaps

gmaps.configure(api_key="AIzaSyDFOjxJ23DfYRLTqEuNsgnqwP0E79Aybpk")
```

```
# weights is passed to describe the imports of each row
locations = dataset[["Latitude", "Longitude"]]
weights_1 = dataset["Accident_Severity"]
fig = gmaps.figure()
fig.add_layer(gmaps.heatmap_layer(locations, weights=weights_1))
fig
```

```
from ipywidgets.embed import embed_minimal_html

embed_minimal_html("export0.html", views=[fig])
```

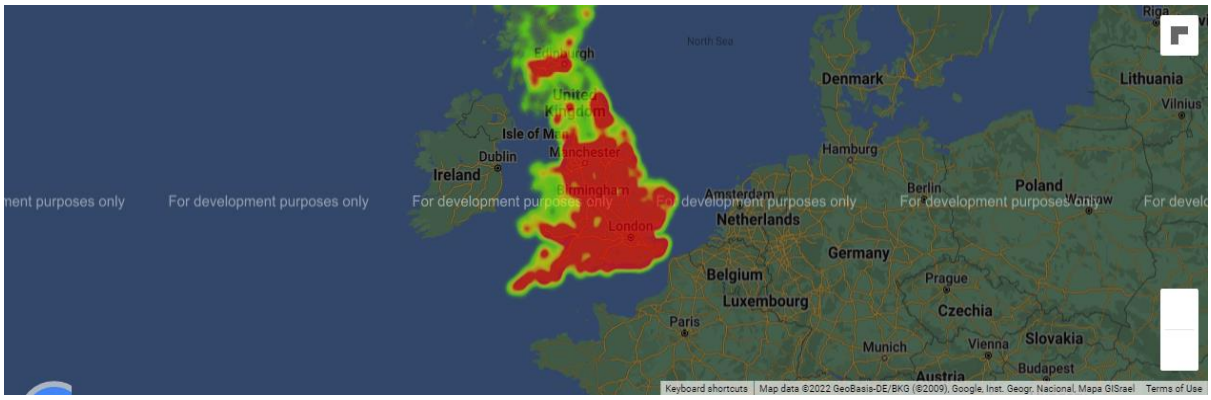


Figure 23: Map of Accidents UK

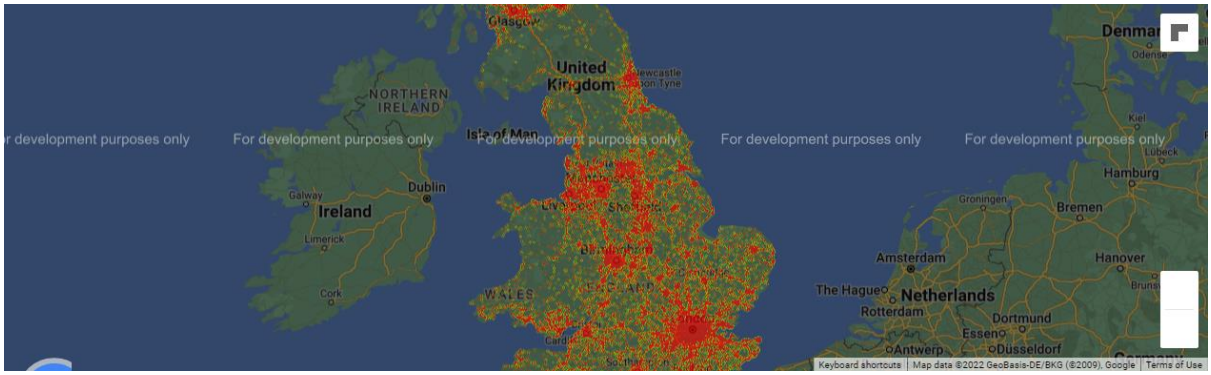


Figure 24: Map of Accident Hotspots

Some events pull crowd along these axes.

The most frequent accident location from data is M60, woodhouses, M43 7ZP, United Kingdom - a major highway in Manchester which is close to Manchester United Stadium and Etihad Stadium. Below, the coordinates of old Trafford was generated and its distance from accident hotspot.

```
from geopy.geocoders import Nominatim
geolocator = Nominatim(user_agent="GoogleV3")
location = geolocator.reverse("53.500209, -2.127225")
print(location.address)

print((location.latitude, location.longitude))

print(location.raw)
```

```
M60, Daisy Nook, Oldham, Woodhouses, Greater Manchester, North West England, England, M43 7ZP, United Kingdom
(53.50019582677351, -2.1272410117431444)
```

```
# Search radius
rad = 8

dist = find_location("Old Trafford, Manchester \n\n\n", rad)
print("\n")
print("Four cordinates to signify the search radius -%s" % dist)
```

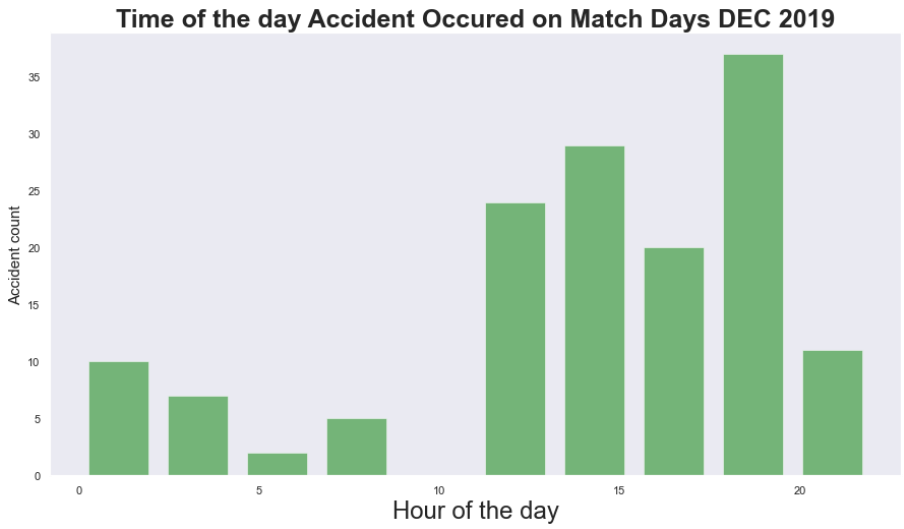
```
Old Trafford, Sir Matt Busby Way, Wharfside, Gorse Hill, Trafford, Greater Manchester, North West England, England, M16 0RA, Un
ited Kingdom
(53.46310955, -2.2913864850545362)
```

```
from geopy.distance import geodesic

M60_manchester = (53.50019582677351, -2.1272410117431444)
Old_trafford = (53.46310955, -2.2913864850545362)
print(geodesic(M60_manchester, Old_trafford).miles)
```

```
7.240674606712318
```

On matchdays 2019, data suggests that the area experienced 5419 accidents at afternoon kick-off and evening kickoff, and severity was most non-injurious.



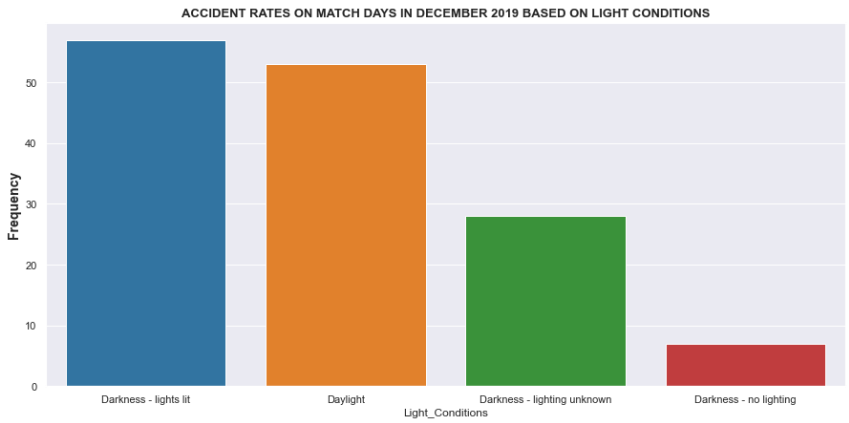


Figure 25: Light Conditions (Matchdays)

Light and weather conditions did not play significant role in the accidents on matchdays.

75% of vehicle involved in accidents on matchdays are cars, average age <= 35, road surface condition mostly wet but significant number also dry.

Weather and Light Conditions:

Most accidents occur during daylight, and a fine weather.

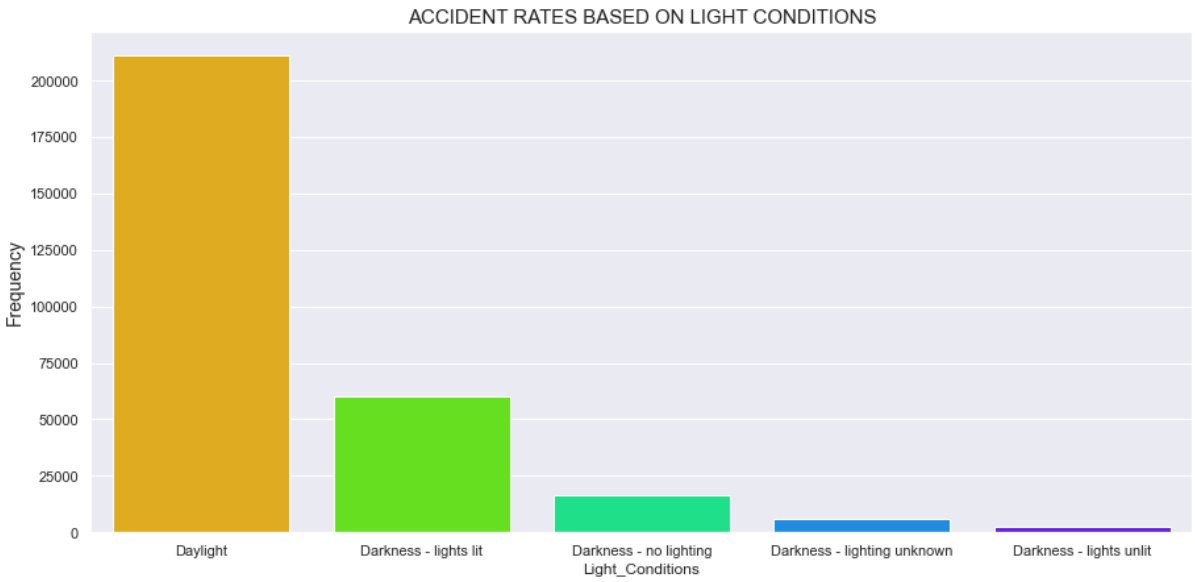


Figure 26: Light Conditions

Accident Geographic Clustering:

We used the Kmeans Elbow method to identify hotspots(clusters) where frequent accidents occur.

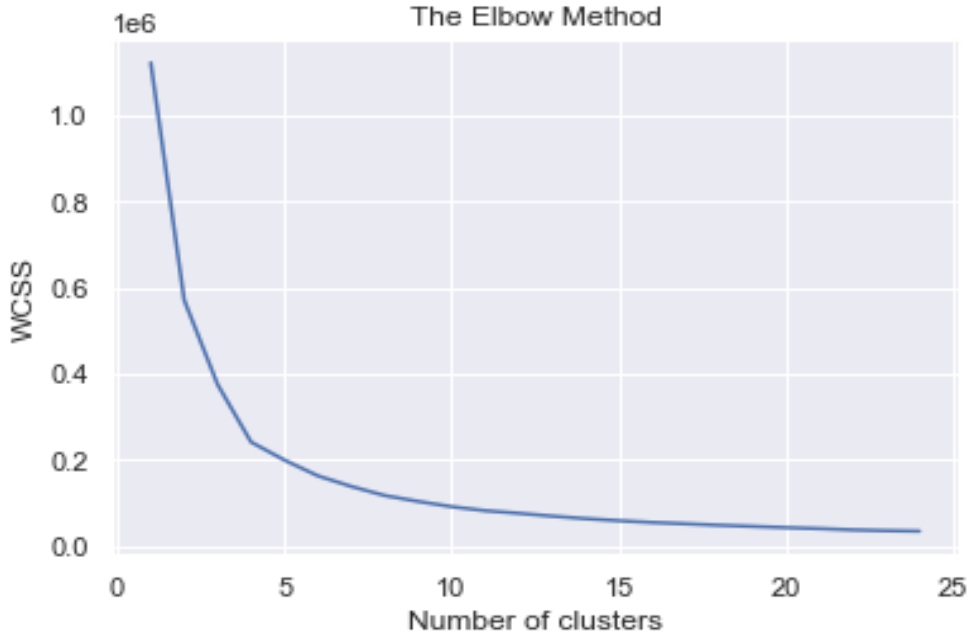


Figure 27: KMeans Elbow Method

The diagram suggests London, Bristol, Birmingham, Manchester, Glasgow area of the United Kingdom

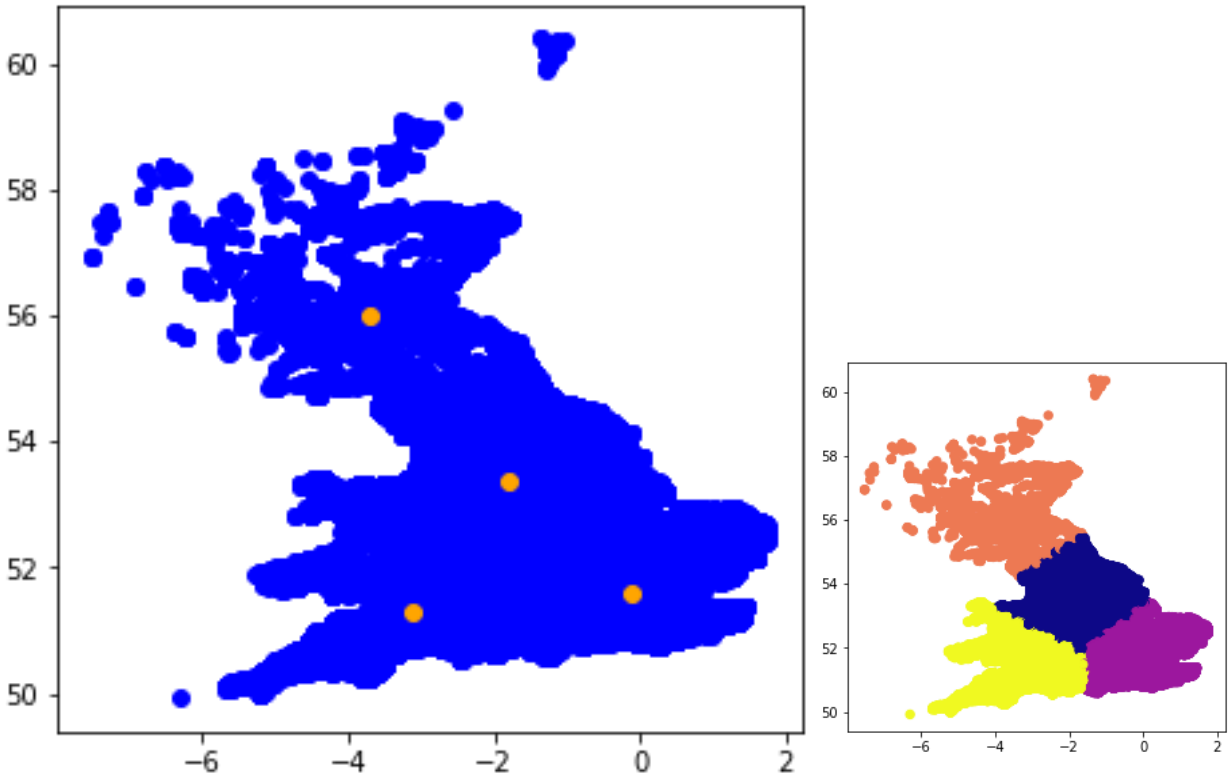


Figure 28: Geographic Clustering

age band of drivers involved in most accidents as below is 26-35, journey of purpose unknown and sex of driver male.

(h) How does driver related variables affect the outcome?

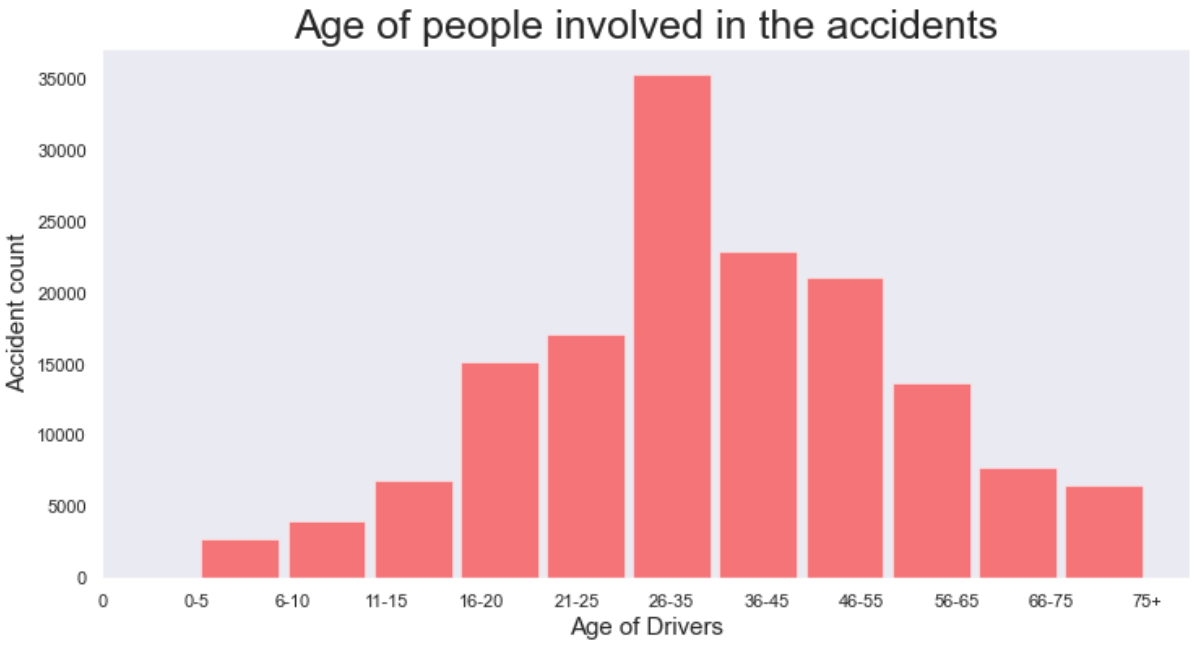


Figure 29: Age Band of Driver

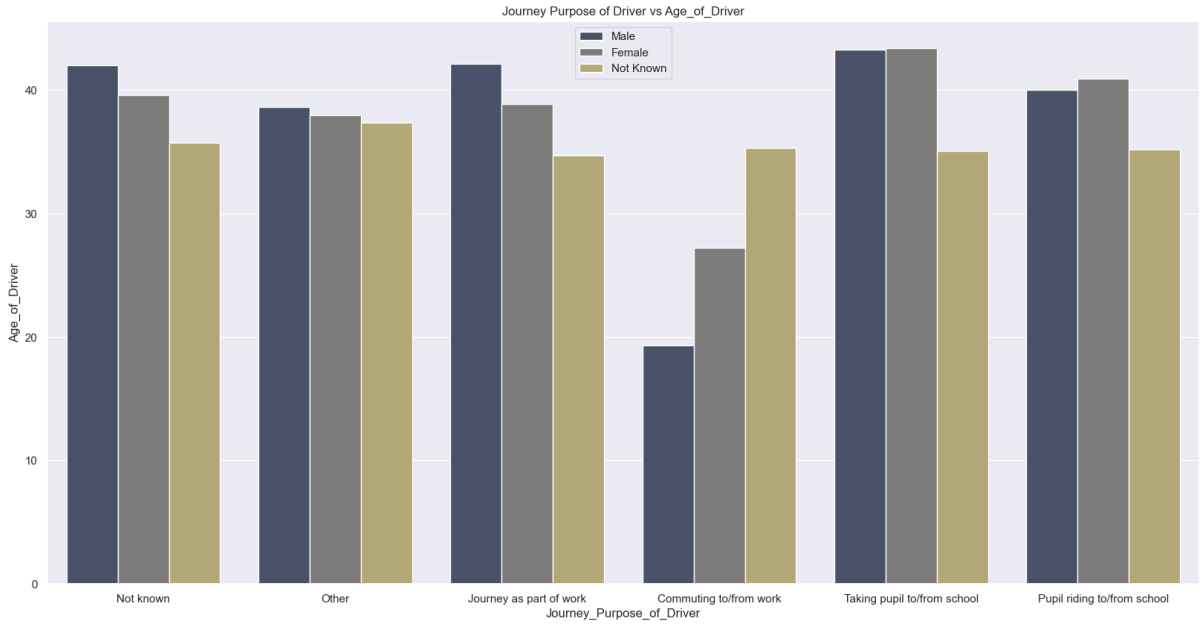


Figure 30: Journey of Purpose of Driver

Hypothesis Testing

Using Apriori to create a deduction of principle, the first hypothesis produced 15 rules that revealed that most seniors with journey of purpose undefined will likely be involved in accident with confidence 30% average. The second hypothesis that if age of driver is 55, Male, and journey of purpose not known, there is >50 confidence that accident will occur which slightly but does not completely support what the data reveals that most age of driver likely to be involved in accidents is 26-35, journey of purpose unknown and sex Male.

	Left Hand Side	Right Hand Side	Support	Confidence	Lift
0	12	4	0.000328	0.267218	82.879245
1	13	4	0.000301	0.241848	75.010639
2	3	35	0.083771	0.800550	5.701960
3	71	5	0.001627	0.322170	2.102831
4	74	5	0.001407	0.349874	2.283656
5	75	5	0.001083	0.306807	2.002557
6	77	5	0.000988	0.329571	2.151138
7	79	5	0.001147	0.369684	2.412957
8	80	5	0.000788	0.309019	2.016990
9	81	5	0.000765	0.328488	2.144071
10	82	5	0.000724	0.344051	2.245653
11	83	5	0.000687	0.343486	2.241959
12	84	5	0.000616	0.346667	2.262722
13	86	5	0.000403	0.329640	2.151587
14	87	5	0.000365	0.339623	2.216746
15	88	5	0.000318	0.354717	2.315268

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedents_len
0	(1)	(6)	0.256467	0.527901	0.208670	0.813630	1.541257	0.073280	2.533134e+00	1
1	(6)	(1)	0.527901	0.256467	0.208670	0.395282	1.541257	0.073280	1.229553e+00	1
2	(1)	(55)	0.256467	0.671262	0.256459	0.999969	1.489685	0.084303	1.068018e+04	1
3	(55)	(1)	0.671262	0.256467	0.256459	0.382055	1.489685	0.084303	1.203235e+00	1
4	(6)	(55)	0.527901	0.671262	0.527866	0.999934	1.489632	0.173506	4.964037e+03	1
5	(55)	(6)	0.671262	0.527901	0.527866	0.786378	1.489632	0.173506	2.209973e+00	1
6	(1, 6)	(55)	0.208670	0.671262	0.208670	1.000000	1.489731	0.068598	inf	2
7	(1, 55)	(6)	0.256459	0.527901	0.208670	0.813655	1.541304	0.073284	2.533475e+00	2
8	(6, 55)	(1)	0.527866	0.256467	0.208670	0.395308	1.541359	0.073289	1.229606e+00	2
9	(1)	(6, 55)	0.256467	0.527866	0.208670	0.813630	1.541359	0.073289	2.533322e+00	1
10	(6)	(1, 55)	0.527901	0.256459	0.208670	0.395282	1.541304	0.073284	1.229566e+00	1
11	(55)	(1, 6)	0.671262	0.208670	0.208670	0.310861	1.489731	0.068598	1.148289e+00	1

Prediction

Architecture 1:

48 attributes were selected for the ensembled modeling. Data was split into 20% test, scaled and trained using Logistic Regression, Linear Discriminant Analysis, Decision Tree Classifier, Random Forest Classifier, and Naïve Bayes. Random Forest had the highest training accuracy of 85.9% and test score 86.3%. The model was a reasonably good fit.

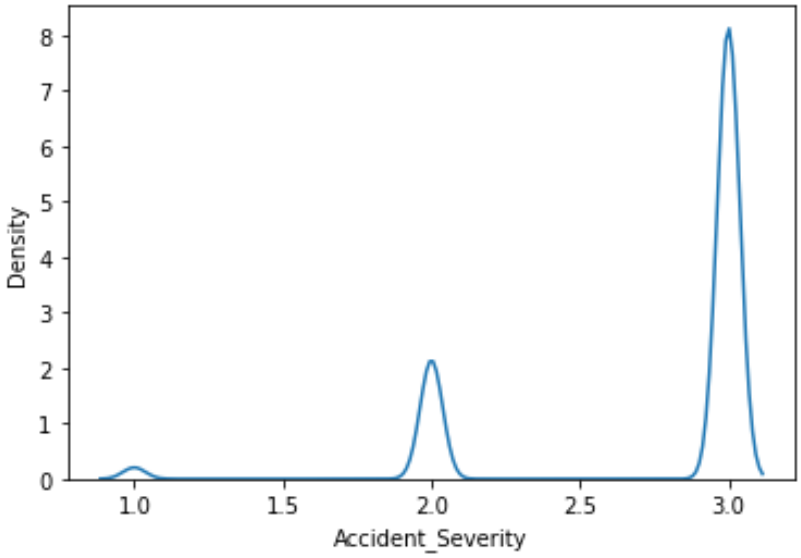


Figure 31: Before Oversampling

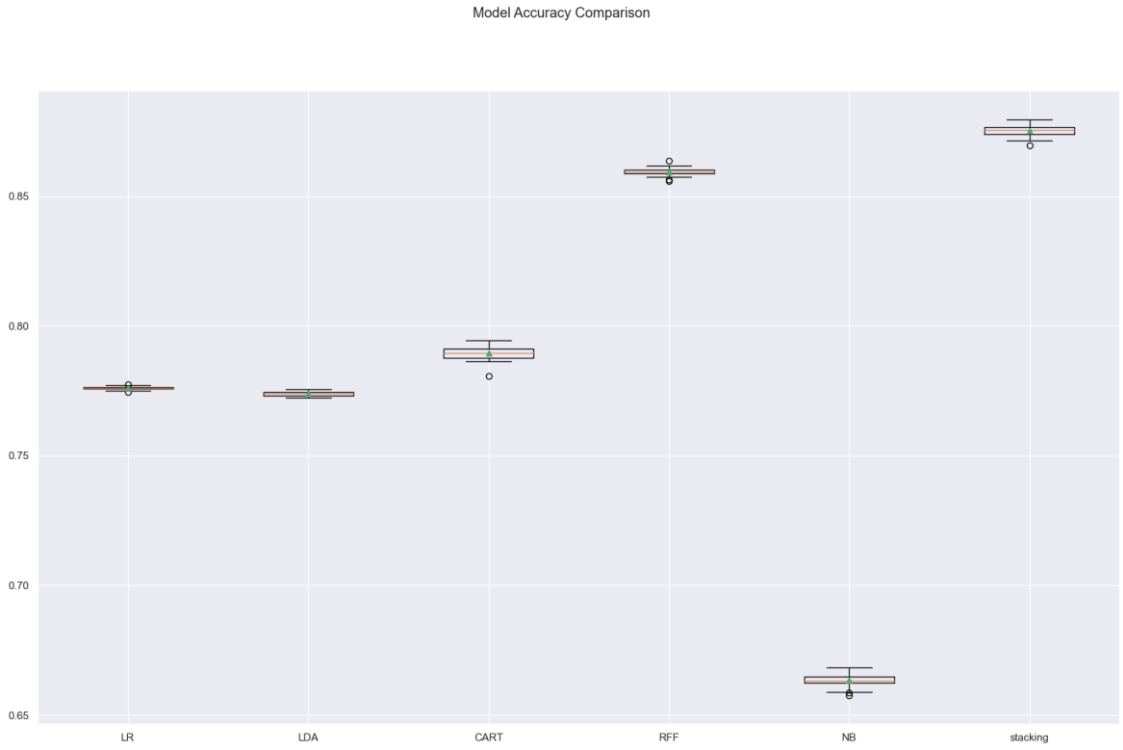


Figure 32: Model Accuracy Comparison

Architecture 2:

Feature engineering using Principal Component Analysis selected easting and Northing columns with explained variance of 0.80976773 0.19004158.

Result: 86% train, 86.4% test.

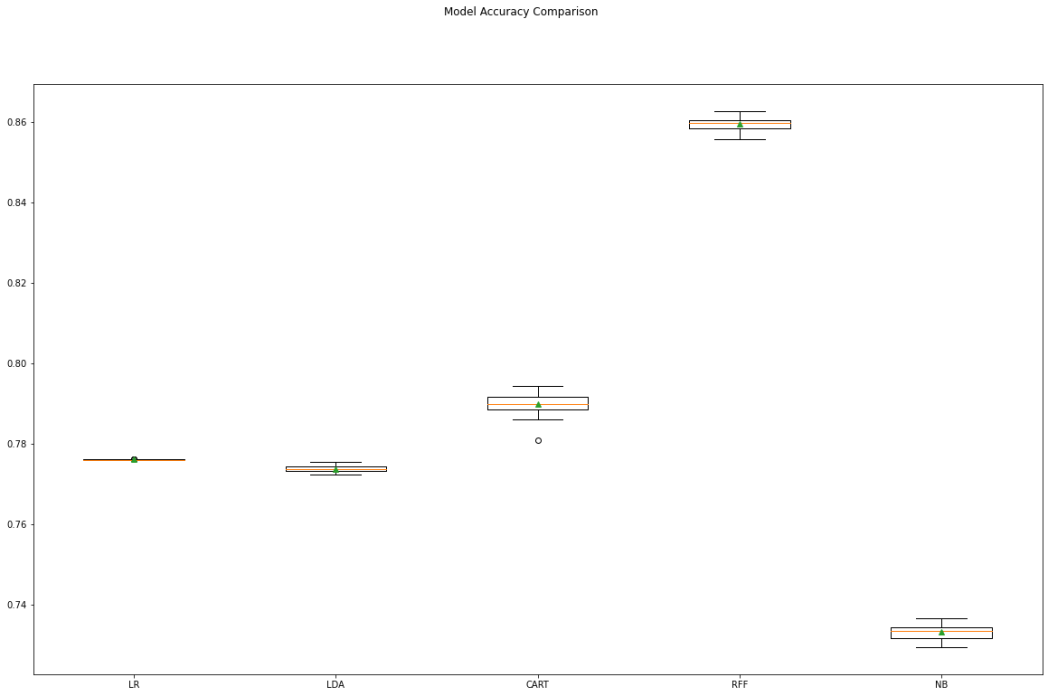


Figure 33: Model Accuracy Comparison (PCA)

Architecture 3:

Model was fed with fewer features (15).

Result = 87.44% test

Comparing Model with CAS Adjustment Data

1. We filtered all the 2019 accident from cas adjustment data using the Accident index as our unique identifier.

```
cas_adjustment = pd.read_csv(
    "cas_adjustment_lookup_2019.csv", dtype={"Accident_index": int}
)
cas_adjustment.head()
```

	accident_index	Vehicle_Reference	Casualty_Reference	Adjusted_Serious	Adjusted_Slight	Injury_Based
0	200401BS00001	1	1	0.239280	0.760720	0
1	200401BS00002	1	1	1.000000	0.000000	0
2	200401BS00003	1	1	0.057141	0.942859	0
3	200401BS00003	1	2	0.048599	0.951401	0
4	200401BS00004	1	1	0.187000	0.813000	0

```
acc_index = dataset["Accident_Index"].tolist()
```

```
cat = cas_adjustment[(cas_adjustment["Accident_Index"].isin(acc_index))]
```

```
cat
```

	Accident_Index	Vehicle_Reference	Casualty_Reference	Adjusted_Serious	Adjusted_Slight	Injury_Based
3163331	2019010128300	1	2	0.000000	1.000000	1
3163332	2019010128300	1	1	0.000000	1.000000	1
3163333	2019010128300	1	3	0.000000	1.000000	1
3163334	2019010152270	1	1	0.000000	1.000000	1
3163335	2019010155191	2	1	0.000000	1.000000	1
...	...	...	...	...	...	...
3314732	2019984106919	1	1	0.183336	0.816664	0
3314733	2019984107019	1	1	0.074588	0.925412	0

2. Predict.proba generated our predictions as shown in the snippet below. We then use our r2_score to compare these continuous results.

```
Y_pred_prob = RFF.predict_proba(X_test)
Y_pred_prob
```

```
array([[0.016, 0.416, 0.568],
       [0.012, 0.42 , 0.568],
       [0.012, 0.42 , 0.568],
       ...,
       [0.028, 0.384, 0.588],
       [0.028, 0.476, 0.496],
       [0.032, 0.408, 0.56 ]])
```

```
# This reveals how the classes were predicted in the array above(fatal, serious, slight)
RFF.classes_
```

```
array([1, 2, 3], dtype=int64)
```

```
pred_prob = pd.DataFrame(
    Y_pred_prob, columns=["Injury_Based", "Adjusted_Serious", "Adjusted_Slight"]
)
```

```
pred_prob
```

```
]: pred_prob
```

]:

	Injury_Based	Adjusted_Serious	Adjusted_Slight
0	0.016	0.416	0.568
1	0.012	0.420	0.568
2	0.012	0.420	0.568
3	0.028	0.432	0.540
4	0.016	0.452	0.532
...	...	...	...
59111	0.044	0.436	0.520
59112	0.016	0.412	0.572
59113	0.028	0.384	0.588
59114	0.028	0.476	0.496
59115	0.032	0.408	0.560

59116 rows × 3 columns

Result:

```
]: from sklearn.metrics import r2_score

r2_score(
    cat, pred_prob,
)
```

```
]: -1.1379837366353656
```

Architecture 4:

Oversampling

We used Smote library to balance our label for better accuracy.

```
]: from imblearn.over_sampling import SMOTE, RandomOverSampler

   sm = SMOTE(random_state=2)
   XX_train, YY_train = sm.fit_resample(XX, YY.ravel())
```

```
]: print("After OverSampling, the shape of train_X: {}".format(XX_train.shape))
   print("After OverSampling, the shape of train_y: {} \n".format(YY_train.shape)

   After OverSampling, the shape of train_X: (688068, 48)
   After OverSampling, the shape of train_y: (688068,)
```

```
]: print("label '1' count after oversampling: {}".format(sum(YY_train == 1)))
   print("label '2' count after oversampling: {}".format(sum(YY_train == 2)))
   print("label '3' count after oversampling: {}".format(sum(YY_train == 3)))

   label '1' count after oversampling: 229356
   label '2' count after oversampling: 229356
   label '3' count after oversampling: 229356
```

```
]: new_dataset = XX_train
   new_dataset["Accident Severity"] = YY_train
```

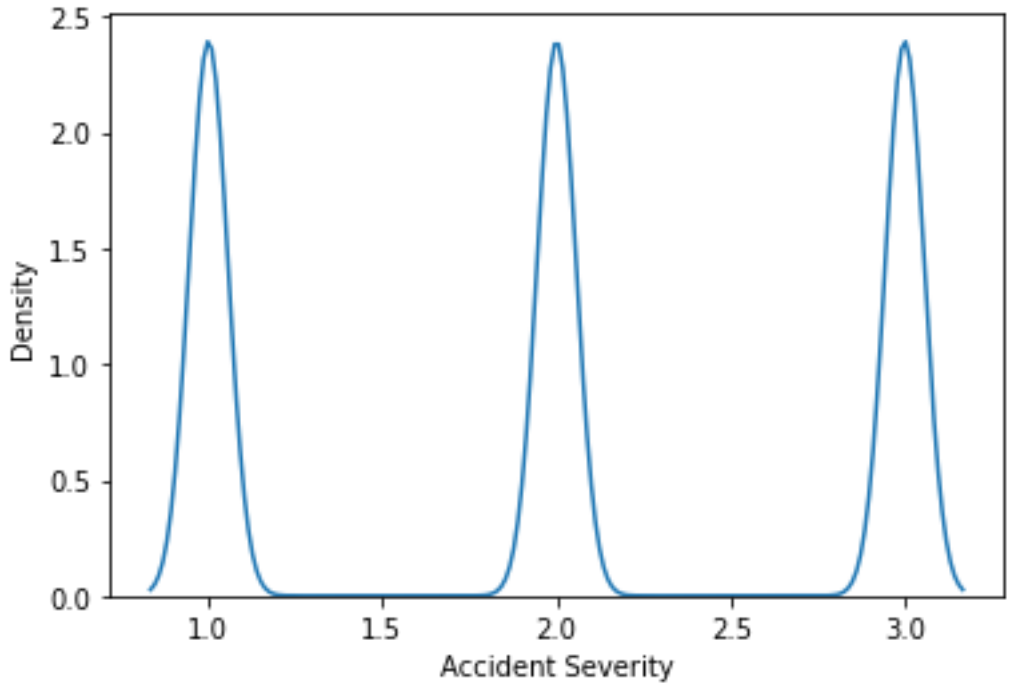


Figure 34: Oversampling

After oversampling, we fit our matrices of features to our model.

Training Accuracy: 100%

Test Accuracy: 100%

```
7]: #Training Accuracy after oversampling

    round(RFF.score(XX_train, YY_train) * 100, 2)
```

```
7]: 100.0
```

```
: from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

RFF = RandomForestClassifier()
RFF.fit(XX_train, YY_train)
XX_pred = RFF.predict(XX_validation)
print("Accuracy: %s%%" % (100 * accuracy_score(YY_validation, XX_pred)))
print(confusion_matrix(YY_validation, XX_pred))
print(classification_report(YY_validation, XX_pred))
```

```
Accuracy: 100.0%
[[ 1188     0     0]
 [    0 11959     0]
 [    0     0 45969]]
              precision    recall  f1-score   support

           1       1.00      1.00      1.00      1188
           2       1.00      1.00      1.00     11959
           3       1.00      1.00      1.00     45969

    accuracy                           1.00     59116
   macro avg       1.00      1.00      1.00     59116
weighted avg       1.00      1.00      1.00     59116
```

R2_score after oversampling:

```
from sklearn.metrics import r2_score

r2_score(
    cat.values, new_proba.values,
)
```

```
-1.444384714435155
```

Recommendations

Government should:

- build pathways for bikers on farm roads experiencing high accident rates.
- encourage wearing motorcycle helmet correctly.
- define more road drainages channels to remove water from road surfaces quickly.
- better crowd management and increase road safety personnel on matchdays around old Trafford and Etihad stadiums.
- policy change to disallow alcohol intake before football match and at half time (Sky Sports, 2021).

Bibliography

Arrive Alive, 2022. *Safe Driving at Sunrise and Sunset / Dusk and Dawn.*
Available at: https://www.arrivealive.mobi/safe-driving-at-sunrise-and-sunset-dusk-and-dawn
[Accessed 20 March 2022].

GeoPy, 2018. *GeoPy.*
Available at: https://geopy.readthedocs.io/en/stable/
[Accessed 31 March 2022].

GOV.UK, 2020. *National Statistics - Reported road casualties Great Britain, annual report: 2019.*
Available at: https://www.gov.uk/government/statistics/reported-road-casualties-great-britain-annual-report-2019#:~:text=Details,of%205%25%20compared%20to%202018
[Accessed 24 April 2022].

GOV.UK, 2022. *Collection - Road accidents and safety statistics.*
Available at: https://www.gov.uk/government/collections/road-accidents-and-safety-statistics
[Accessed 24 April 2022].

Ministry of Housing, Communities & Local Government, 2018. *Lower Layer Super Output Area in England.*
Available at:
https://opendatacommunities.org/atlas/resource?uri=http%3A%2F%2Fopendatacommunities.org%2Fid%2Fgeography%2Fadministration%2Flsoa%2FE01031267
[Accessed 5 April 2022].

Pypi, 2020. *Bidirectional UTM-WGS84 converter for python.*
Available at: https://pypi.org/project/utm/0.4.2/#description
[Accessed 20 March 2022].

Sky Sports, 2021. *Drinking in seats at football grounds could be reinstated as part of a fan-led review into the game.*
Available at: https://skysports.com/football/news/11095/12416189/drinking-in-seats-at-football-grounds-could-be-reinstated-as-part-of-a-fan-led-review-into-the-game#:~:text=Currently%2C%20drinking%20alcohol%20is%20permitted,and%20quickly%20at%20half-time.
[Accessed 29 April 2022].

Truscello, D. M., 2019. *Sun Glare and Car Accidents.*
Available at: https://www.truscellolaw.com/blog/2019/06/sun-glare-car-accidents/
[Accessed 20 March 2022].