Simple Neural Networks to Predict Accident Severity in the UK

Group 07: Rajat, Ashly, Suphanat(Jeep), Avish, Sunwha, Vinul

November 18, 2018

Our Work

Initial Thoughts

After setting up a structure for our day, we broke up into smaller groups to split up the work more efficiently. We soon realised that a major part of this task was cleaning the data up to make it analysable. After finishing our respective tasks, we would come together to combine our work.

The data was quite overwhelming to begin with. There were 70 different variables (columns) and around 300,000 entries (rows) for a total of about 2.1 million cells. The variables were split up into 'Vehicle', data related to the vehicle and driver; 'Casualty', data related to people injured in the accident; and 'Accident Circumstances', any details not related to the vehicle or casualties. 'Vehicle', and 'Casualty.' An important part of this project was finding out something to measure and so this was the first task for the group.

The initial plan was doing a time series over an extended period of time to measure the likelihood of car accidents. The goal was to statistically determine the important factors which make crashes more likely - perhaps to work with people who could improve these factors in a real-world situation. This idea evolved over the course of the progress and was made more general to finding the likelihoods of casualty severity based on some of the accident circumstances. These were chosen to be longitude and latitude, and a time series. If we had more time, more variables would be plotted.

Data Clearing

Data Clearing was fundamental to the whole process. The data needed to be in a format that could be analysed easily. The team realised that null data values were represented by different numbers (and sometimes strings) for some variables. Code needed to be written that recognised these 'pseudo-null' (which we defined as not int or floats) and 'null' values, then changed them to 'np.nan' - the null value for a pandas *dataframe*. Pandas was used instead of numpy due to its wider functionality.

A brief check on the columns showed that there were plenty of null values we had to delete. The brute force method was not only inefficient, but unable to locate all the 'null' values. Another method was created which deleted a column from the data if over 40% of its values were null. This only removed five columns out of 70 so another method was required.

For some accidents, there are two or more vehicles involved in an accident. Looking through the data, for vehicles that has the same accident indices (i.e. were in the same accident) only one of the vehicles would have the data completely filled out for all the variables and the others had their last 14 variables set to null. To clean up the data, these accidents were located and the vehicles with null parameters were deleted. This gave us only unique values. This refined data was easier to process and stopped our overall trend being weighted more towards cases where more than one vehicle is involved.

Initially there were two different formattings of time, one of them was in D/M/Y format and the other was an unclear string value. Later in the project, it was discovered that the latter date was specific Excel date string which counts the number of days from 1900 and then gives the time i.e. it converts a string to integer. In order to simplify these, all the dates were converted to the Excel format. The conversion was done this way round to to make future calculations and eventual graph plotting easier by only using numbers.

The correlation between variables was determined using the function `corr()` from the pandas library. We removed the graphs with correlation above 0.9 since they both had the same trends and so would not affect the overall trend of the results. This also helped to remove noise and make the graphs more readable.

A *Random Forest Classifier (RFC)* was used to fill in null values. The RFC generates lots of different decision trees which use different variables from our initial data set. Each row could have a number of missing values, so we looked at all the categories that could have missing values. We then ran a RFC for each of these categories, ultimately, to run these models on each row to fill in these missing values. It is essentially an advanced *Map-Reduce* algorithm. SciKit Learn functions were used for this task. In the data when we were filling in blanks, sometimes there were continuous variables rather than discrete variables and so they required a *Regression model* alongisde to a Classifier to predict them. The advantage of using a Random Forest instead of a normal Decision Tree is that you multiple smaller decision trees where each of them get a subset of the features that you have and so they all make slightly different predictions. This avoids *overfitting*.

The final step of data cleaning was removing outliers to avoid *underfitting*. The way we did was finding columns which had continuous data and then removed any rows which had data that was above or below two standard deviations from the mean. The heuristic we used to deem whether feature was continuous was whether it had more than 70 unique values in its column. We realised that longitude and latitude came under this and so factored them out.

Principal Component Analysis (PCA)

Theory

This technique helps to reduce a higher-dimensionality problem to a lower-dimensional one - also known as a *dimensional reduction.* It first identifies the *hyperplane* that lies closest to the data (essentially a fancy way of saying the plane that best fits the data), and then it projects the data onto it.

To identify this hyperplane, you need to do planes of best fit (called axes) in different angles and figure out which of these keep the greatest variance of the data. It is important to project from axes which reflect the greatest variance in data because these will lose less information than other projections. This line is the first Principle Component (PC). We then find (d-1) other PCs (where d was our initial number of dimensions) from the lines that are orthogonal to the first PC. These orthogonal PCs account for the largest remaining variances in the training set.

Practical Application

This technique is coded through matrix operations. From an input data of covariances between each dimension (for us, a a square matrix of dimensions corresponding to the number of variables we use), the PCA matrix creates a *diagonal matrix* of variances for the each orthogonal axis. Each matrix entry on the diagonal is a fraction of the overall sum on the diagonal.

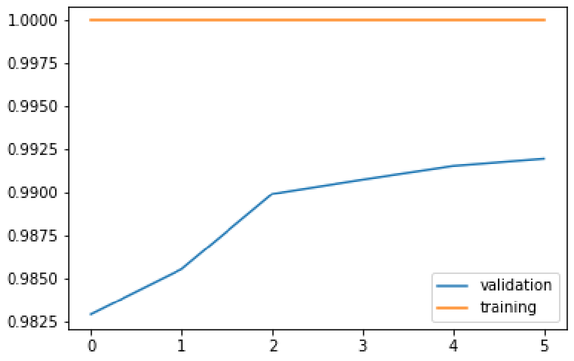
After this a matrix factorisation technique called *Singular Value Decomposition (SVD)* was used to break down the training set matrix into three other matrices. The final of these matrices contained all the principal components that were needed. The training set was then projected onto the hyperplane using another matrix calculation. A batch-wise PCA can be used to gradually develop the model and introduces an element of randomness to the model. The optimal hyper parameter for this was 44 as shown in the graph below. We chose to use SciKit Learn's toolkit because it provided all of these functions and centred the data after projections.

The Model

A *neural network with dropout* was built to predict whether the severity of an accident will be 1, 2 or 3, by classifcation. We used dropout to reduce overfitting with an optimal dropout rate of 0.8. An optimal gradient descent optimiser was selected through trial and error. We picked the momentum optimiser with Nestorov-accelerated gradient, which adds 'friction' to our gradient descent to prevent entering local minima and result in a smoother descent which you can see in our loss convergence graph in Appendix.

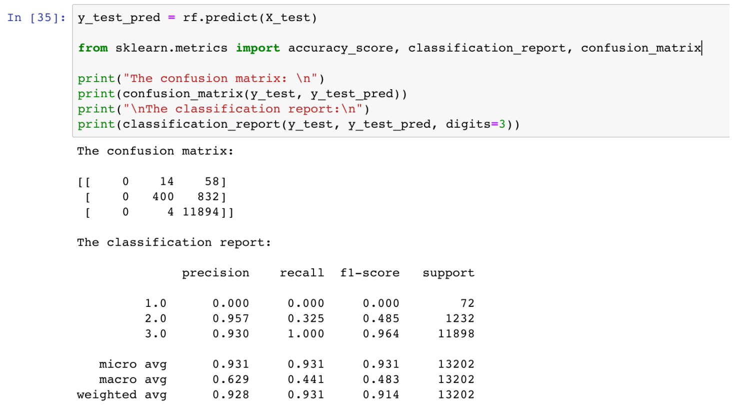
The following were optimised for training.

Hyperparameters	Optimal parameters after tuning
$N_HiddenLayer1$	300
$N_HiddenLayer2$	100
$\eta(LearningRate)$	0.1
$\mu(Momentum)$	0.9
Epoch	6
Batch Size	100



Further Analysis

We intended to do a timeseries analysis using an Recurrent Neural Network (RNN) but the cleaning of the dataset meant that this wasn't feasible. Instead we ended up using a RFC from SciKit Learn to factor in how a change of time affects the severity once again. We merged our initially cleaned up data to another pandas dataframe with the dates in sequential order. Our classifier gave us the following confusion matrix.

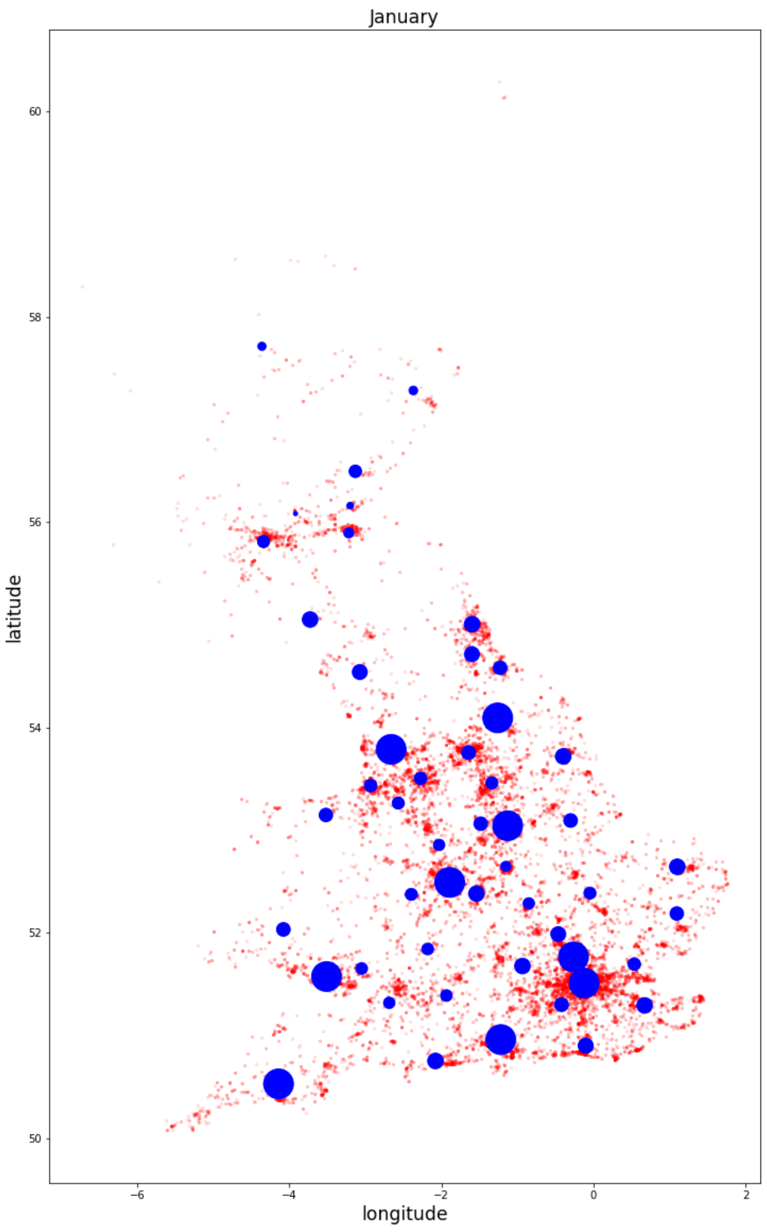


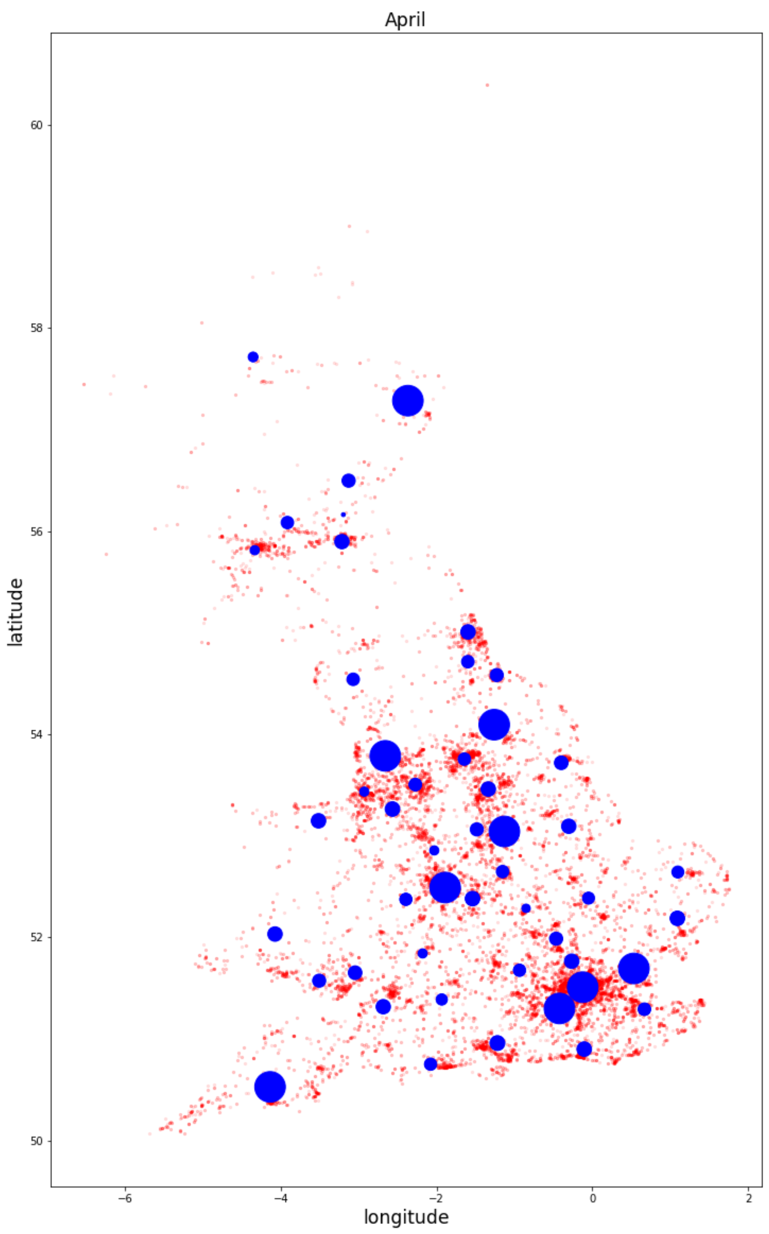
```
In [35]: y_test_pred = rf.predict(X_test)

         from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

         print("The confusion matrix: \n")
         print(confusion_matrix(y_test, y_test_pred))
         print("\nThe classification report:\n")
         print(classification_report(y_test, y_test_pred, digits=3))

The confusion matrix:

[[     0     14     58]
 [     0    400    832]
 [     0      4  11894]]

The classification report:

              precision    recall  f1-score   support

         1.0      0.000     0.000     0.000        72
         2.0      0.957     0.325     0.485      1232
         3.0      0.930     1.000     0.964     11898

   micro avg      0.931     0.931     0.931     13202
   macro avg      0.629     0.441     0.483     13202
weighted avg      0.928     0.931     0.914     13202
```

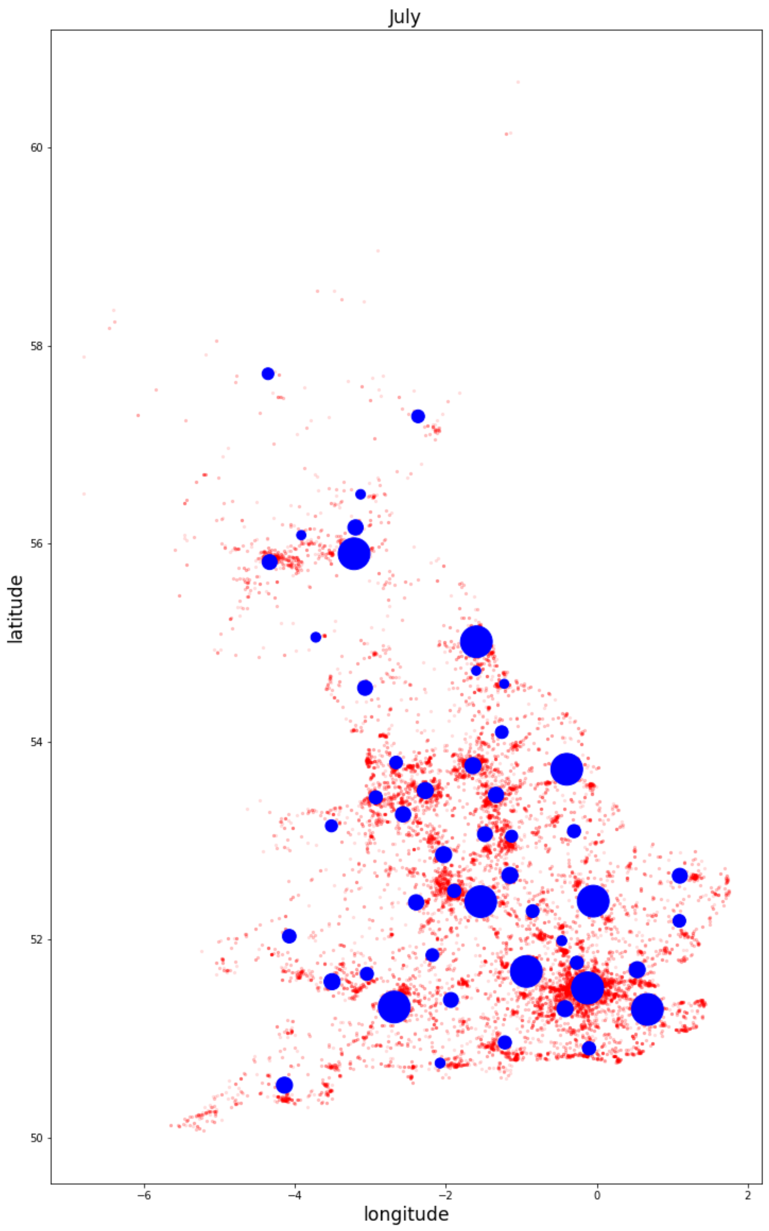
Glossary

Classifier: Used to investigate the relationship between two or more variables and estimate one variable based on the others. Model for predicting **discrete data**.
Dataframe: A class from the Python library `pandas` that has useful functions for arrays. *Diagonal Matrix*: A matrix which only has values along its diagonal, all other values are zero. This means that for entries D_m,n, there are only values when m = n. *Dimensional Reduction*: Reducing a higher-dimensional problem to a lower dimensional one e.g. going from a matrix with 50 dimensions to one with three - often to make visualisations and comparisons easier.
Hyperplane: Plane that best fits the data - think of this as a line of best fit extended to higher dimensions.
Map-Reduce: The general name for an algorithm that maps an initial data set to another set, then reduces the set down to one value. *Neural Network with Dropout*: A neural network is a network of weighted nodes which we train to make predictions. The dropout referes to deciding when to turn off neurons within layers in our network based off of a given probability. *Overfitting*: When you've trained a model so well on the training set that you are unable to make further predictions. *Principal Component*: An axis that is most reflective of a variance (spread) of data.
Principal Component Analysis: Specific technique for Dimensional Reduction (defined in these key terms).
Random Forest Classifier: Algorithm that generates lots of decision trees to develop a model. Used on discrete data.
Regression Model: Used to investigate the relationship between two or more variables and estimate one variable based on the others. Model for predicting **continuous data**.
Underfitting: A model doesn't fit the training set well.

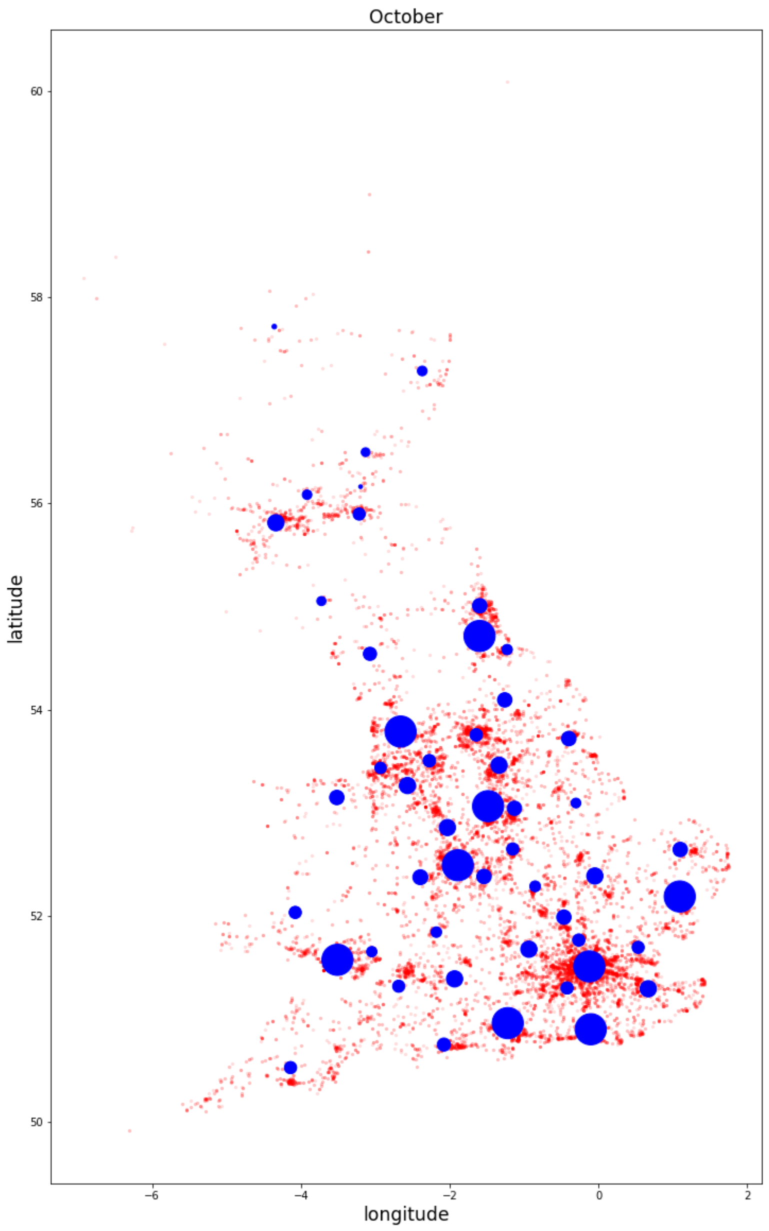
Appendix

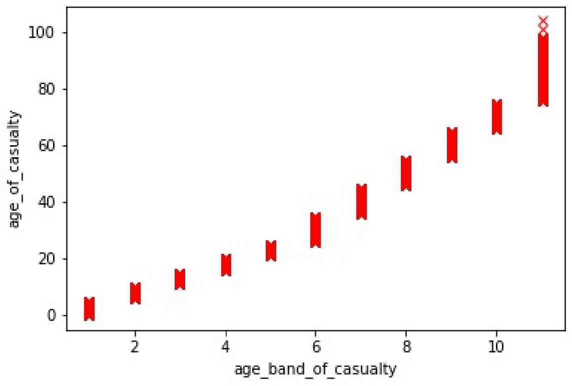


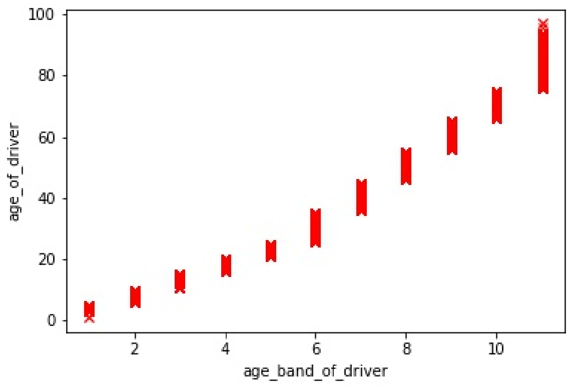

April
latitude
longitude
60
58
56
54
52
50
−6
−4
−2
0
2

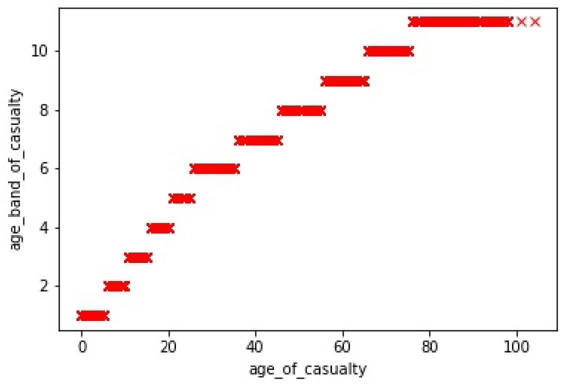


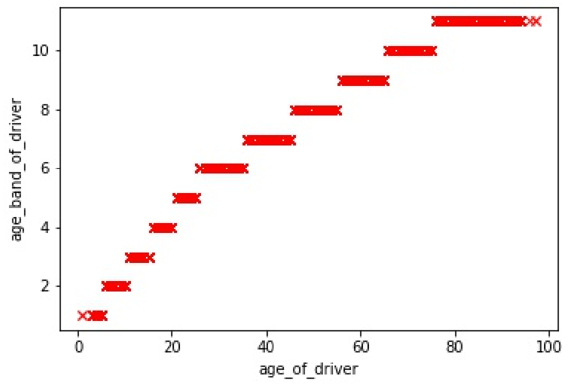
July
60
58
56
54
52
50
latitude
−6
−4
−2
0
2
longitude

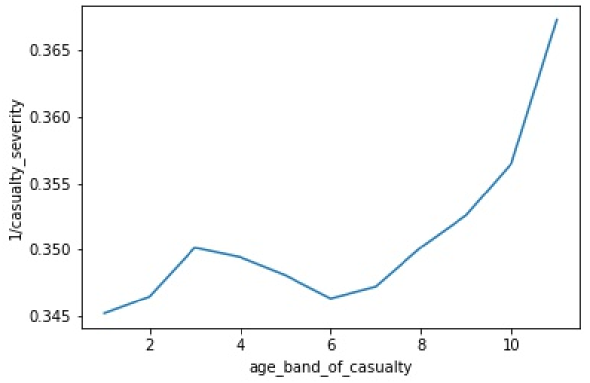

October
latitude
longitude
60
58
56
54
52
50
−6
−4
−2
0
2

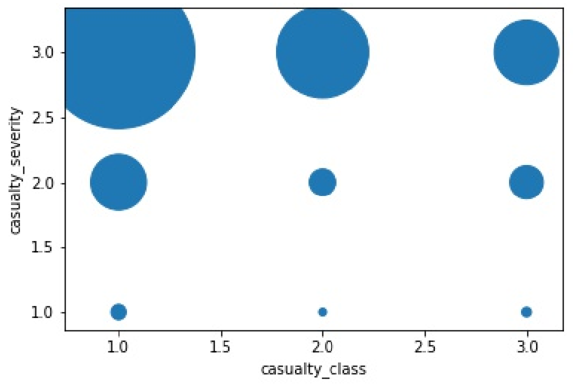

age_of_casualty
age_band_of_casualty
0
20
40
60
80
100
2
4
6
8
10

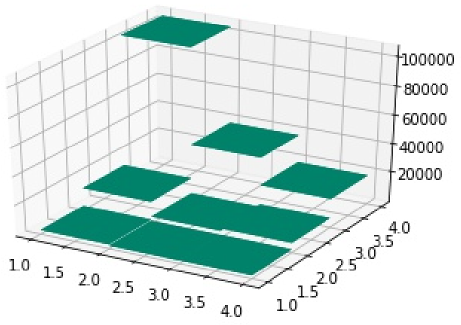

age_of_driver
age_band_of_driver
0
20
40
60
80
100
2
4
6
8
10

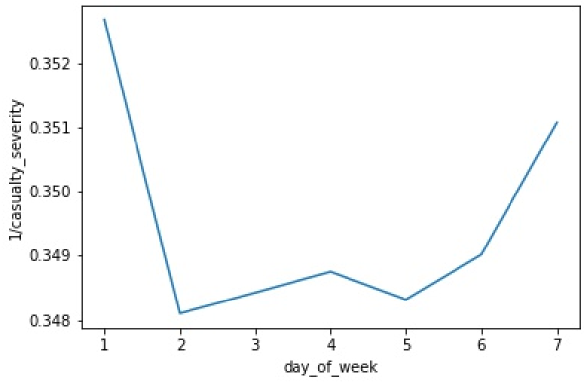

age_band_of_casualty
age_of_casualty
2
4
6
8
10
0
20
40
60
80
100

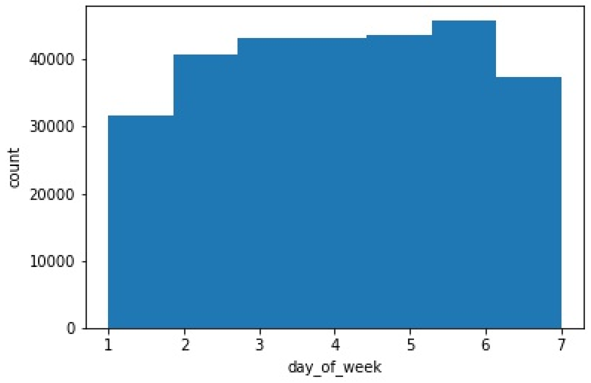

age_band_of_driver
age_of_driver
0
20
40
60
80
100
2
4
6
8
10

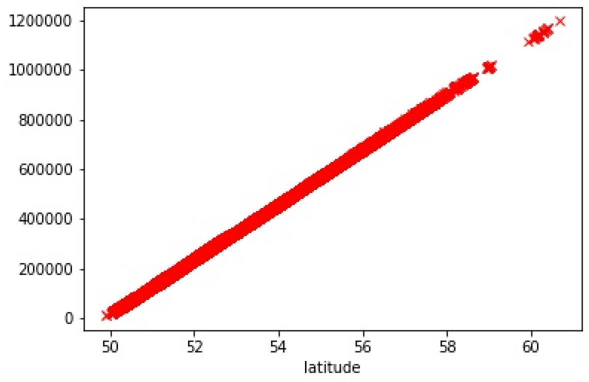

1/casualty_severity
age_band_of_casualty
0.345
0.350
0.355
0.360
0.365
2
4
6
8
10


casualty_severity
casualty_class
1.0
1.5
2.0
2.5
3.0

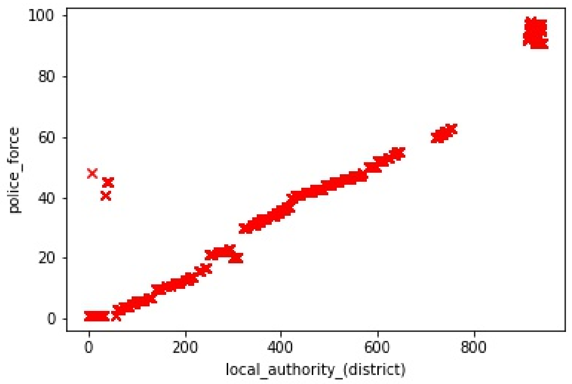

100000
80000
60000
40000
20000
1.0
1.5
2.0
2.5
3.0
3.5
4.0
1.0
1.5
2.0
2.5
3.0
3.5
4.0


1/casualty_severity
0.352
0.351
0.350
0.349
0.348
1
2
3
4
5
6
7
day_of_week

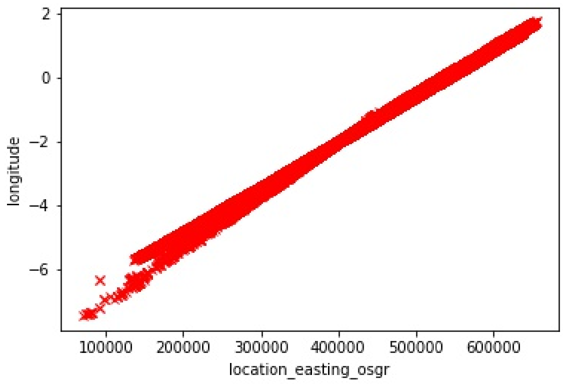

count
40000
30000
20000
10000
0
1
2
3
4
5
6
7
day_of_week



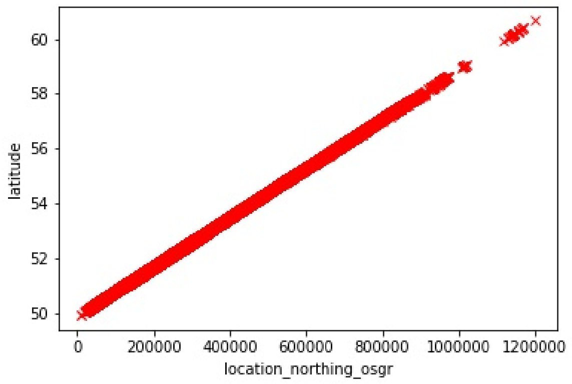
1200000
1000000
800000
600000
400000
200000
0
50
52
54
56
58
60
latitude

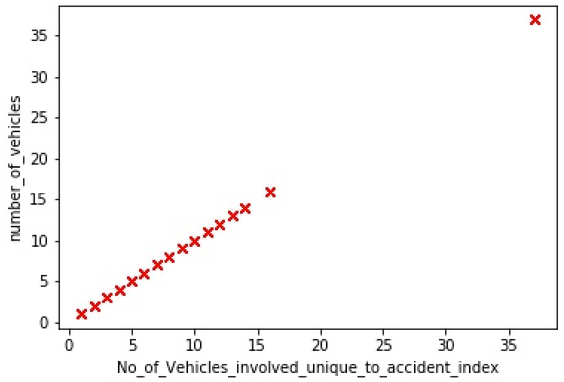


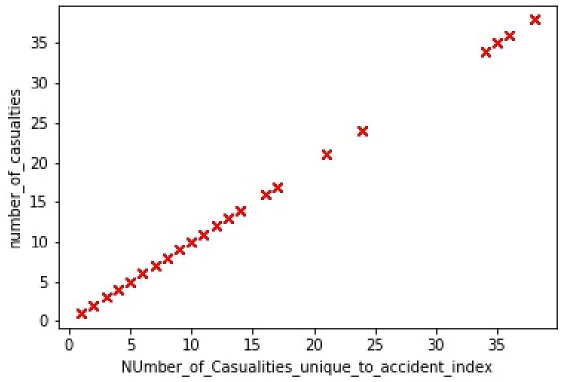
100
80
60
40
20
0
police_force
0
200
400
600
800
local_authority_(district)

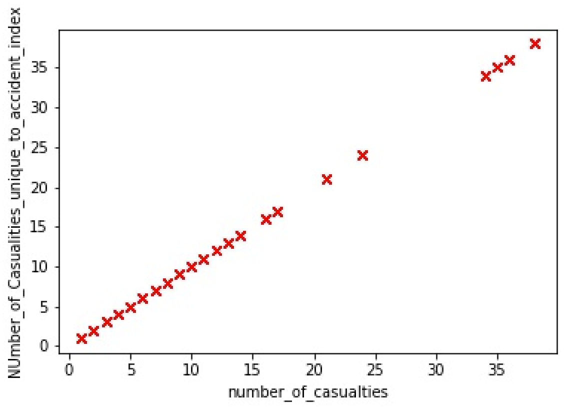


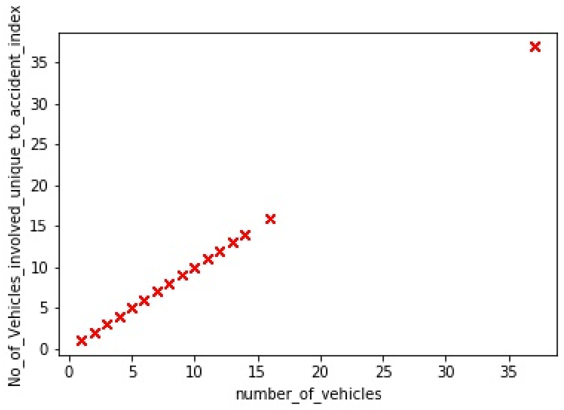
2
0
−2
−4
−6
longitude
100000
200000
300000
400000
500000
600000
location_easting_osgr

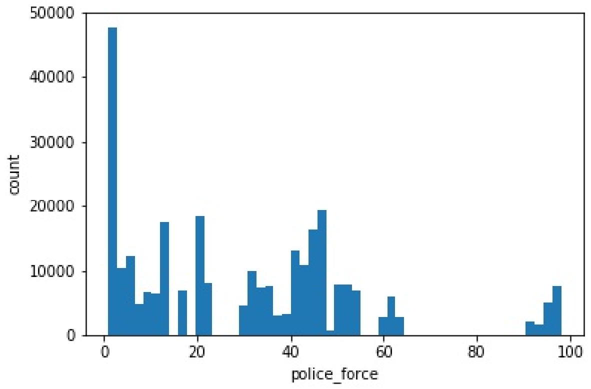

latitude
location_northing_osgr
50
52
54
56
58
60
0
200000
400000
600000
800000
1000000
1200000

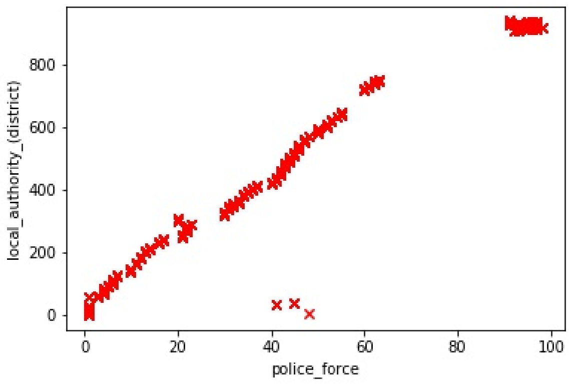

number_of_vehicles
No_of_Vehicles_involved_unique_to_accident_index
0
5
10
15
20
25
30
35


number_of_casualties
NUmber_of_Casualities_unique_to_accident_index
0
5
10
15
20
25
30
35


NUmber_of_Casualities_unique_to_accident_index
number_of_casualties


No_of_Vehicles_involved_unique_to_accident_index
number_of_vehicles


count
police_force


local_authority_(district)
police_force
0
200
400
600
800
0
20
40
60
80
100