Configuration Manual

MSc Research Project
Data Analytics

Dhruv Vimal Shah
Student ID: X21121087

School of Computing
National College of Ireland

Supervisor: Qurrat Ul Ain

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Dhruv Vimal Shah
Student ID:	X21121087
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Qurrat Ul Ain
Submission Due Date:	15/12/2022
Project Title:	Prediction of Accident Severity Using Machine Learning Algorithms
Word Count:	1129
Page Count:	16

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Dhruv Vimal Shah
Date:	1st February 2023

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Dhruv Vimal Shah
X21121087

1 Introduction

This setup manual provides a summary of the prerequisites, both in terms of software and hardware, to reproduce the research. This manual will help you understand the coding methods needed to do this study again, from setting up the environment to looking at the model's results. The following is a detailed instructional guide that has been broken up into numerous sections for your convenience.

2 Environmental Setup

This section contains a list of all of the tools and software that were used to complete the project successfully.

2.0.1 Hardware Requirements

The hardware specifications used for this project were a 64-bit Windows 10 operating system and 8GB of RAM. The processor used was an Intel i7 (8th Gen). Figure 1 shows the details of the hardware specifications used.

2.0.2 Software Requirements

Python is the programming language that was utilised for the development of the models since it is capable of scripting and executing machine learning models within a web browser. Jupyter Notebook, version 6.4.5, which is supported by Anaconda was used to carry out the code's execution. Because the system is 64-bit compatible the first step is to install the Anaconda application. The link to download the application can be found here [1]. Following successful installation, the dashboard will appear as illustrated in Figure 2. Once anaconda has been installed just click on launch Jupyter notebook and it would be opened and ready to code.

3 Importing Libraries

There are some libraries that need to be installed from the 'pip' command. The installation is done as "pip(library_name)" at the anaconda environment's command prompt. And there are some libraries that are pre-installed in the Anaconda navigator. So to import them just the command mentioned in Figure 3 is required.

[1]Anaconda Download: `http://www.Anaconda.com/downloads`

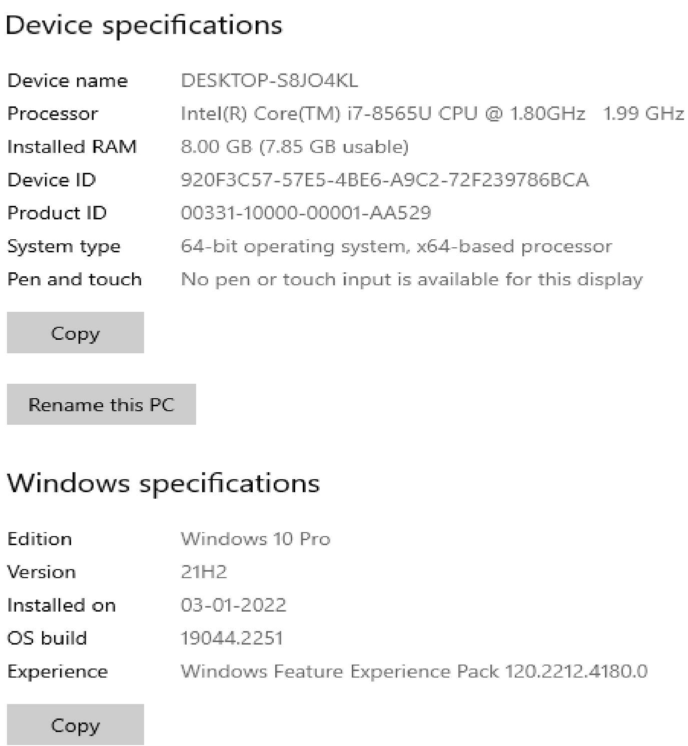


Figure 1: Hardware and Windows Specification

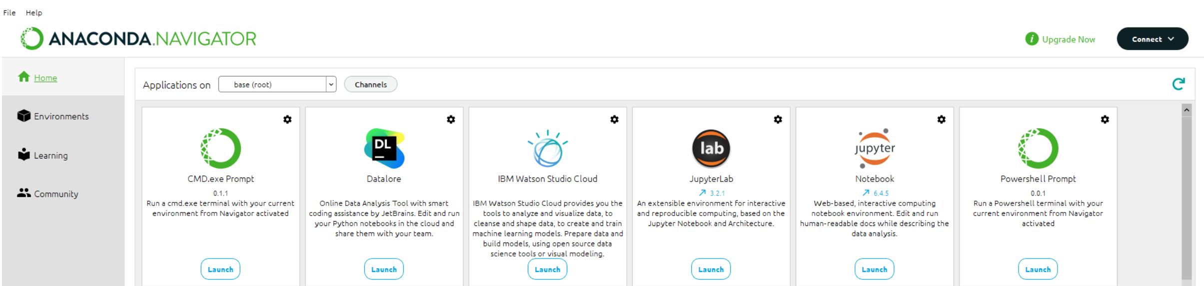


Figure 2: Anaconda Home Page Dashboard

Importing Libraries

```
In [63]: import datetime as dt
         from datetime import datetime
         import pandas as pd
         import matplotlib.pyplot as plt
         import numpy as np
         import seaborn as sns
         from mpl_toolkits.basemap import Basemap
         from sklearn.model_selection import TimeSeriesSplit
         plt.style.use('ggplot')
         %config InlineBackend.figure_format = 'retina'
         import warnings
         warnings.filterwarnings('ignore')

         import folium
         from folium.plugins import HeatMap
         import matplotlib.pyplot as plt
         plt.ticklabel_format(useOffset=False)

         %matplotlib inline
         import math
         import statsmodels.api as sm
         from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LogisticRegression
         from sklearn.tree import DecisionTreeClassifier
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.svm import SVC, LinearSVC
         from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
         from sklearn.metrics import roc_auc_score,roc_curve,f1_score,recall_score,precision_score
```

Figure 3: Importing Library

4 Datasets

The data selected was the "UK Car Accidents 2005-2015" dataset from Kaggle [2]. The data was initially extracted from the United Kingdom's Department of Transport. It contains 3 files (Accident0515, Casualties0515, and Vehicles0515) in (.csv) format. Accident0515 is the main file, and through the Accident Index column, it has links to Casualties0515 and Vehicles0515. The Accident0515 file comprises 1780653 rows and 31 columns. The Casualties0515 file contains 2216720 rows and 14 columns, and the Vehicles0515 file has 3004425 rows and 21 columns. Figure 4 shows the code for importing all these files and seeing their shapes.

Importing Data

```
In [47]: accidents = pd.read_csv('Accidents0515.csv',index_col='Accident_Index')
         casualties=pd.read_csv('Casualties0515.csv' , error_bad_lines=False,index_col='Accident_Index',warn_bad_lines=False)
         vehicles=pd.read_csv('Vehicles0515.csv', error_bad_lines=False,index_col='Accident_Index',warn_bad_lines=False)

In [49]: accidents.shape
Out[49]: (1780653, 31)

In [97]: casualties.shape
Out[97]: (681643, 14)

In [98]: vehicles.shape
Out[98]: (3004425, 21)
```

Figure 4: Importing Data

[2]Dataset: https://www.kaggle.com/datasets/silicon99/dft-accident-data

5 Data Exploring

The data were explored to get some insight which could be helpful to create a good model. Figures 5, 6, and 7 show the data exploration in the research. It includes how many values are there in a particular column and even finding the sum of null values.

```
In [133]: accidents.head() #Displays the 1st 5 records in the accident file.
```

Out[133]:

Accident_Index	Location_Easting_OSGR	Location_Northing_OSGR	Longitude	Latitude	Police_Force	Accident_Severity	Number_of_Vehicles	Number_of_Casu
200501BS00001	525680.0	178240.0	-0.191170	51.489096	1	2	1	
200501BS00002	524170.0	181650.0	-0.211708	51.520075	1	3	1	
200501BS00003	524520.0	182240.0	-0.206458	51.525301	1	3	2	
200501BS00004	526900.0	177530.0	-0.173862	51.482442	1	3	1	
200501BS00005	528060.0	179040.0	-0.156618	51.495752	1	3	1	

5 rows × 31 columns

```
In [134]: accidents.describe()
```

Out[134]:

	Location_Easting_OSGR	Location_Northing_OSGR	Longitude	Latitude	Police_Force	Accident_Severity	Number_of_Vehicles	Number_of_Casual
count	1.780515e+06	1.780515e+06	1.780515e+06	1.780515e+06	1.780653e+06	1.780653e+06	1.780653e+06	1.780653e
mean	4.401799e+05	2.985128e+05	-1.428579e+00	5.257450e+01	3.075348e+01	2.838219e+00	1.832064e+00	1.349454e
std	9.547599e+04	1.612254e+05	1.403495e+00	1.451753e+00	2.551787e+01	4.017948e-01	7.148732e-01	8.237479
min	6.495000e+04	1.029000e+04	-7.516225e+00	4.991294e+01	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e
25%	3.764000e+05	1.779900e+05	-2.354786e+00	5.148769e+01	7.000000e+00	3.000000e+00	1.000000e+00	1.000000e
50%	4.413200e+05	2.649500e+05	-1.386472e+00	5.227209e+01	3.100000e+01	3.000000e+00	2.000000e+00	1.000000e
75%	5.234235e+05	3.965700e+05	-2.164865e-01	5.346409e+01	4.600000e+01	3.000000e+00	2.000000e+00	1.000000e
max	6.555400e+05	1.208800e+06	1.762010e+00	6.075754e+01	9.800000e+01	3.000000e+00	6.700000e+01	9.300000e

8 rows × 27 columns

Figure 5: Description of the data

6 Cleaning the data

After the exploration of the data now the unwanted columns such as Location_Easting_OSGR, Location_Northing_OSGR, LSOA_of_Accident_Location, Junction_Control, and 2nd_Road_Class being removed from the data as it has a very high number of null values in it. Figure 8 shows the code for it.

7 Exploratory Data Analysis (EDA)

For a better understanding of the data, the EDA process is carried out using the variables of the dataset. This process explains what kind of data is present in the dataset. Different

```
In [135]: accidents.info() #Prints information about the Dataframe

<class 'pandas.core.frame.DataFrame'>
Index: 1780653 entries, 200501BS00001 to 2015984141415
Data columns (total 31 columns):
 #   Column                                       Dtype
---  ------                                       -----
 0   Location_Easting_OSGR                        float64
 1   Location_Northing_OSGR                       float64
 2   Longitude                                    float64
 3   Latitude                                     float64
 4   Police_Force                                 int64
 5   Accident_Severity                            int64
 6   Number_of_Vehicles                           int64
 7   Number_of_Casualties                         int64
 8   Date                                         object
 9   Day_of_Week                                  int64
 10  Time                                         object
 11  Local_Authority_(District)                   int64
 12  Local_Authority_(Highway)                    object
 13  1st_Road_Class                               int64
 14  1st_Road_Number                              int64
 15  Road_Type                                    int64
 16  Speed_limit                                  int64
 17  Junction_Detail                              int64
 18  Junction_Control                             int64
 19  2nd_Road_Class                               int64
 20  2nd_Road_Number                              int64
 21  Pedestrian_Crossing-Human_Control            int64
 22  Pedestrian_Crossing-Physical_Facilities      int64
 23  Light_Conditions                             int64
 24  Weather_Conditions                           int64
 25  Road_Surface_Conditions                      int64
 26  Special_Conditions_at_Site                   int64
 27  Carriageway_Hazards                          int64
 28  Urban_or_Rural_Area                          int64
 29  Did_Police_Officer_Attend_Scene_of_Accident  int64
 30  LSOA_of_Accident_Location                    object
dtypes: float64(4), int64(23), object(4)
memory usage: 434.7+ MB
```

Figure 6: Prints information about the Dataframe

```
In [137]: accidents.isnull().sum()

Out[137]: Location_Easting_OSGR                          138
          Location_Northing_OSGR                         138
          Longitude                                      138
          Latitude                                       138
          Police_Force                                     0
          Accident_Severity                                0
          Number_of_Vehicles                               0
          Number_of_Casualties                             0
          Date                                             0
          Day_of_Week                                      0
          Time                                           151
          Local_Authority_(District)                       0
          Local_Authority_(Highway)                        0
          1st_Road_Class                                   0
          1st_Road_Number                                  0
          Road_Type                                        0
          Speed_limit                                      0
          Junction_Detail                                  0
          Junction_Control                                 0
          2nd_Road_Class                                   0
          2nd_Road_Number                                  0
          Pedestrian_Crossing-Human_Control                0
          Pedestrian_Crossing-Physical_Facilities          0
          Light_Conditions                                 0
          Weather_Conditions                               0
          Road_Surface_Conditions                          0
          Special_Conditions_at_Site                       0
          Carriageway_Hazards                              0
          Urban_or_Rural_Area                              0
          Did_Police_Officer_Attend_Scene_of_Accident      0
          LSOA_of_Accident_Location                   129471
          dtype: int64

In [138]: accidents = accidents.join(vehicles, how='outer')
```

Figure 7: Sum of Null values in the data

```
In [139]: accidents.drop(['Location_Easting_OSGR', 'Location_Northing_OSGR','LSOA_of_Accident_Location',
                         'Junction_Control' ,'2nd_Road_Class'], axis=1, inplace=True)
          accidents['Date_time'] =  accidents['Date'] +' '+ accidents['Time']

          for col in accidents.columns:
              accidents = (accidents[accidents[col]!=-1])

          for col in casualties.columns:
              casualties = (casualties[casualties[col]!=-1])

              accidents['Date_time'] = pd.to_datetime(accidents.Date_time)
          accidents.drop(['Date','Time'],axis =1 , inplace=True)
          accidents.dropna(inplace=True)
```

Figure 8: Data Cleaning

types of histograms are used to explore the different variables. All the EDA done in the research are shown in Figures 9, 10, 11, and 12.

```
In [112]: plt.figure(figsize=(12,6))
          accidents.Date_time.dt.dayofweek.hist(bins=7,rwidth=0.55,alpha=0.5, color= 'blue')
          plt.title('Day of Week Accidents Occured' , fontsize= 20)
          plt.grid(False)
          plt.ylabel('Accident Count' , fontsize = 20)
          plt.xlabel('0 - Sunday ,  1 - Monday  ,2 - Tuesday , 3 - Wednesday , 4 - Thursday , 5 - Friday , 6 - Saturday' , fontsize = 14)
```

Figure 9: Sum of Null values in the data

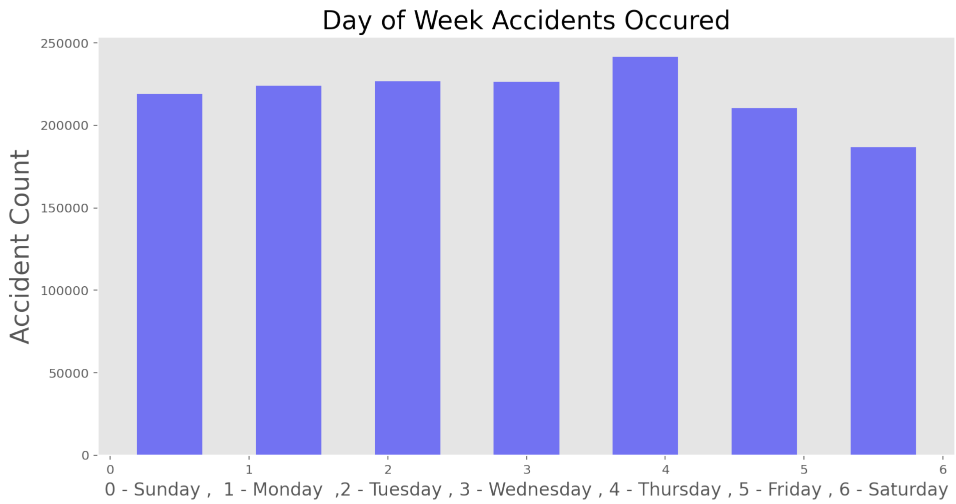


Figure 10: Sum of Null values in the data

8 Plotting Accidents on Open Street Maps

Now we will be using Open Street Maps (OSM) to plot the accidents. Using longitude and latitude information, we can see what area has the most accidents. For using the Open Street Maps the package folium would be required, which was already installed at the start of the code as shown in Figure 3. The accident plots can give us a really good idea about traffic in any area of the UK. Figure 13 depicts the accidents on the map with an "Accident" popup. Also with help of folium the casualties were displayed on the

```
In [111]: plt.figure(figsize=(12,6))
          accidents.Date_time.dt.hour.hist(rwidth=0.75,alpha =0.50, color= 'orange')
          plt.title('Time of the Day/Night',fontsize= 20)
          plt.grid(False)
          plt.xlabel('Time(in Hours)' , fontsize = 20)
          plt.ylabel('Accident Count' , fontsize = 15)

Out[111]: Text(0, 0.5, 'Accident Count')
```

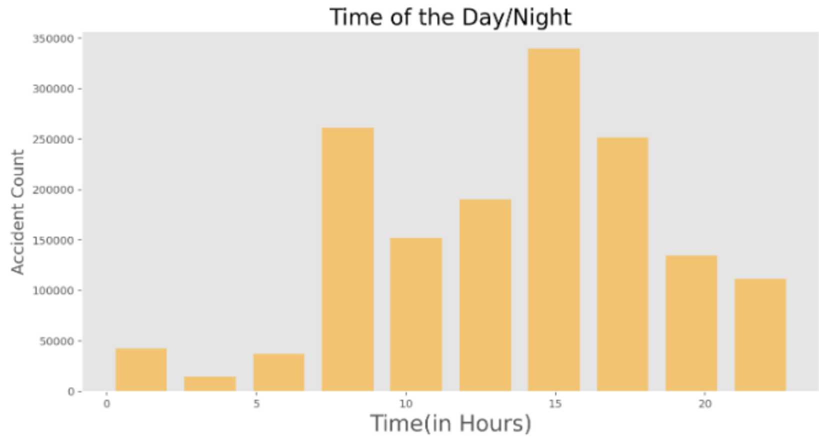


Figure 11: Sum of Null values in the data

```
In [110]: objects = ['0','0-5','6-10','11-15','16-20','21-25','26-35',
                     '36-45', '46-55','56-65','66-75','75+']

          plt.figure(figsize=(12,6))
          casualties.Age_Band_of_Casualty.hist(bins = 11,alpha=0.5,rwidth=0.90, color= 'red',)
          plt.title('People involved in the Accidents (Age Group)', fontsize = 20)
          plt.grid(False)
          y_pos = np.arange(len(objects))
          plt.xticks(y_pos , objects)
          plt.ylabel('Accident Count' , fontsize = 15)
          plt.xlabel('Age of Drivers', fontsize = 15,)

Out[110]: Text(0.5, 0, 'Age of Drivers')
```

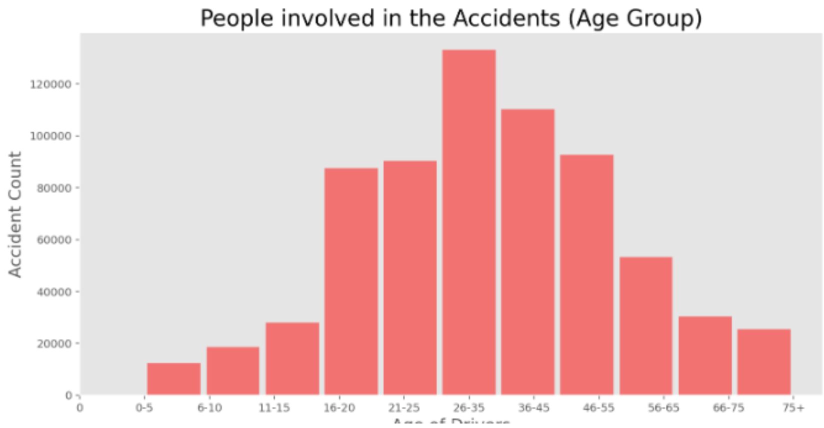


Figure 12: Sum of Null values in the data

map shown in Figure 14. Blue denotes one casualty on the hotspot. Orange denotes 2 casualties on the hotspot and red denotes more than 2 casualties on the given hotspot.

```
location = datad['Latitude'].mean(), datad['Longitude'].mean()
m = folium.Map(location=location,zoom_start=15)

for i in range(0,len(datad)):


    popup = folium.Popup('Accident', parse_html=True)
    folium.Marker([datad['Latitude'].iloc[i],datad['Longitude'].iloc[i]],popup=popup).add_to(m)
m
```

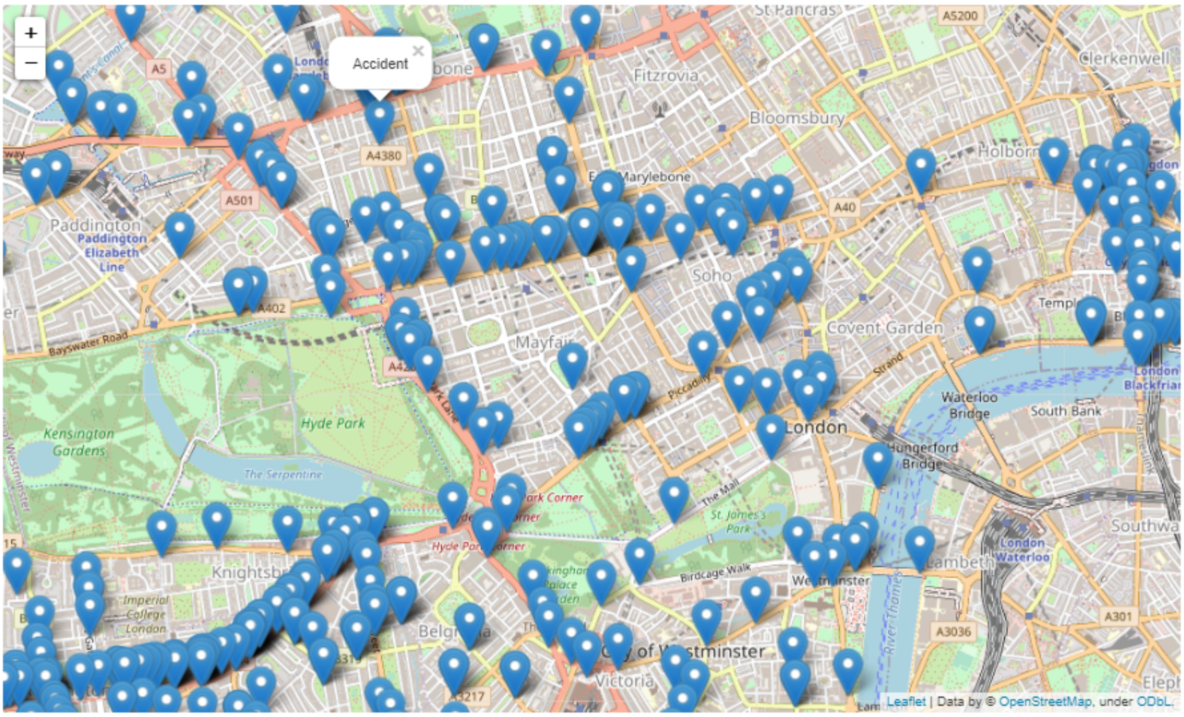


Figure 13: Accidents Plotted on Map

9 Normalization of Data

In this, the normalization of variables was carried out because there were 2 variables that skewed the performance. The variables 'age of driver" and "age of vehicle". Figure 15 shows the before graph and code of both these variables, and Figure 16 shows the after normalization graph of both these variables.

10 Machine Learning

There were 3 machine learning algorithms used in this research. All the algorithms were even tuned with hyperparameters. The section even displays the evaluation of the algorithms.

In [114]:
```python
location = datad['Latitude'].mean(), datad['Longitude'].mean()
m = folium.Map(location=location,zoom_start=15)

#Assigning colour to classify the casuality
for i in range(0,len(datad)):
    num_of_casualties = datad['Number_of_Casualties'].iloc[i]
    if num_of_casualties == 1:
        color = 'blue'
    elif num_of_casualties == 2:
        color = 'orange'
    else:
        color = 'red'
    popup = folium.Popup('Causality', parse_html=True)
    folium.Marker([datad['Latitude'].iloc[i],datad['Longitude'].iloc[i]],popup=popup,icon=folium.Icon(color=color, icon='info-sig

m
```

Out[114]:

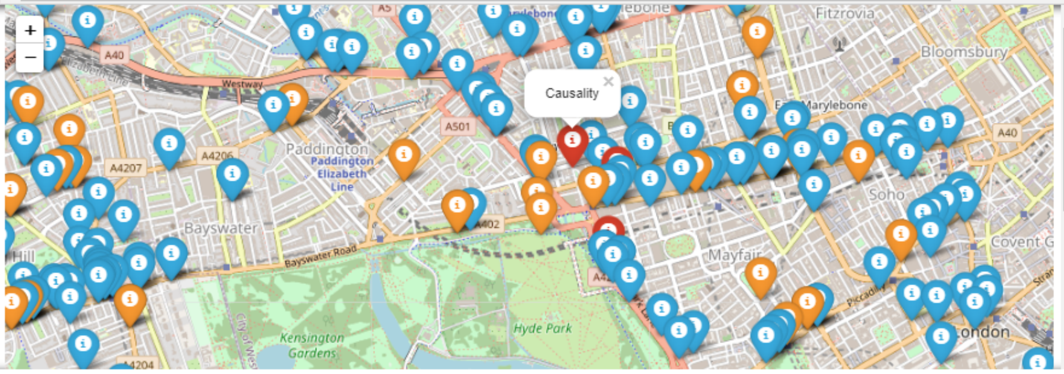
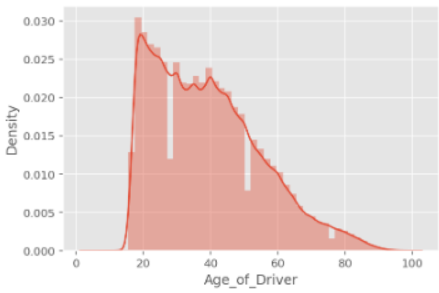


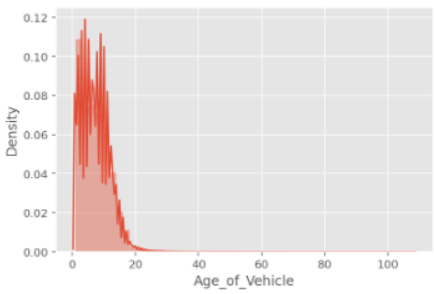
Figure 14: Causalities Plotted on Map

In [20]:
```python
sns.distplot(accidents['Age_of_Driver']);
fig = plt.figure()
```



<Figure size 432x288 with 0 Axes>

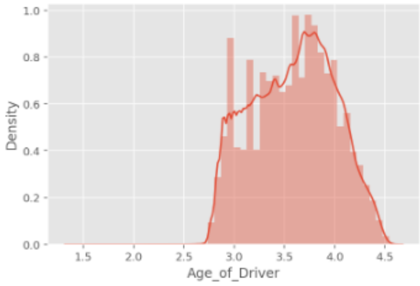
In [21]:
```python
sns.distplot(accidents['Age_of_Vehicle']);
fig = plt.figure()
```



<Figure size 432x288 with 0 Axes>

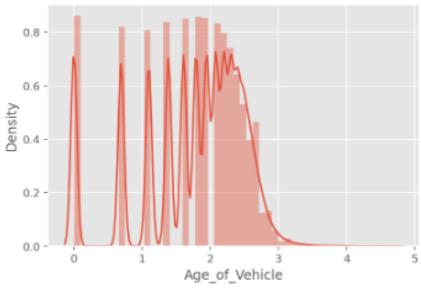
Figure 15: Before normalization

In [22]:
```
accidents['Age_of_Driver'] = np.log(accidents['Age_of_Driver'])
sns.distplot(accidents['Age_of_Driver']);
fig = plt.figure()
```




```
<Figure size 432x288 with 0 Axes>
```

In [23]:
```
accidents['Age_of_Vehicle'] = np.log(accidents['Age_of_Vehicle'])
sns.distplot(accidents['Age_of_Vehicle']);
fig = plt.figure()
```




```
<Figure size 432x288 with 0 Axes>
```

Figure 16: After Normalization

10.1 Train Test Split

To do the training and testing the libraries "from sklearn.model_selection import train_test_split" is required which is shown in Figure 3. The data was divided into 80% training and 20% test data. The random state was kept to 99. Figure 17 shows the train test split.

```
In [25]: accident_ml = accidents.drop('Accident_Severity' ,axis=1)
         accident_ml = accident_ml[['Did_Police_Officer_Attend_Scene_of_Accident' , 'Age_of_Driver' ,'Vehicle_Type', 'Age_of_Vehicle','Eng
                                 , 'Light_Conditions', 'Sex_of_Driver' ,'Speed_limit']]

         # Split the data into a training and test set.
         X_train, X_test, y_train, y_test = train_test_split(accident_ml.values,
                                                   accidents['Accident_Severity'].values,test_size=0.20, random_state=99)
```

Figure 17: Train Test Split

10.2 Random Forest with & without Hyperparameter

For running a random forest model the library required is "sklearn.ensemble import RandomForestClassifier" as shown in Figure 3. Figure 18 shows a random forest model built with default parameters and the classification report which includes the accuracy, precision, recall and f1 score and the confusion matrix. For using this evaluation method the library "from sklearn.metrics import confusion_matrix,accuracy_score,classification_report" and "from sklearn.metrics import roc_auc_score,roc_curve,f1_score,recall_score,precision_score" is used as shown in Figure 3. Figures 19 and 20 show a random forest model built with hyperparameters.

Random Forest

```
In [26]: random_forest = RandomForestClassifier(n_estimators=200)
         random_forest.fit(X_train,y_train)
         Y_pred = random_forest.predict(X_test)
         random_forest.score(X_test, y_test)
         acc_random_forest1 = round(random_forest.score(X_test, y_test) * 100, 2)

         sk_report = classification_report(digits=6,y_true=y_test,y_pred=Y_pred)
         print("Accuracy" , acc_random_forest1)
         print(sk_report)
         pd.crosstab(y_test, Y_pred, rownames=['Actual'], colnames=['Predicted'], margins=True)

Accuracy 84.59
              precision    recall  f1-score   support

           1   0.053114  0.007054  0.012454      4111
           2   0.231720  0.056486  0.090831     38151
           3   0.866542  0.972663  0.916541    264697

    accuracy                       0.845862    306959
   macro avg   0.383792  0.345401  0.339942    306959
weighted avg   0.776748  0.845862  0.801808    306959
```

Out[26]:

Predicted	1	2	3	All
Actual				
1	29	313	3769	4111
2	113	2155	35883	38151
3	404	6832	257461	264697
All	546	9300	297113	306959

Figure 18: Random Forest without hyperparameters

Random Forest Hyperparameter tuning

First, we will see the default parameters of the random forest model before we tune the parameters.

In [31]:
```
random_forest.get_params()
```

Out[31]:
```
{'bootstrap': True,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 200,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

In [36]:
```
from sklearn.model_selection import RandomizedSearchCV
param_grid = {
    'bootstrap': [True],
    'max_depth': [80, 90, 100, 110],
    'max_features': [4, 5],
    'min_samples_leaf': [5, 10, 15],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [100, 200, 300]
}
# Create a based model
random_f = RandomForestClassifier()
# Instantiate the grid search model
grid_search = RandomizedSearchCV(estimator = random_f, param_distributions = param_grid,
                          cv = 3, n_jobs = -1, verbose = 2)

grid_search.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

Out[36]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=-1,
                   param_distributions={'bootstrap': [True],
                                        'max_depth': [80, 90, 100, 110],
                                        'max_features': [4, 5],
                                        'min_samples_leaf': [5, 10, 15],
                                        'min_samples_split': [8, 10, 12],
                                        'n_estimators': [100, 200, 300]},
                   verbose=2)
```

Figure 19: Random Forest with hyperparameters

```
In [37]: Y_pred = grid_search.predict(X_test)
acc_random_forest1 = round(grid_search.score(X_test, y_test) * 100, 2)

sk_report = classification_report(
    digits=6,
    y_true=y_test,
    y_pred=Y_pred)
print("Accuracy" , acc_random_forest1)
print(sk_report)
pd.crosstab(y_test, Y_pred, rownames=['Actual'], colnames=['Predicted'], margins=True)
```

```
Accuracy 86.23
              precision    recall  f1-score   support

           1   0.000000  0.000000  0.000000      4111
           2   0.444380  0.020104  0.038468     38151
           3   0.864674  0.997091  0.926173    264697

    accuracy                       0.862311    306959
   macro avg   0.436351  0.339065  0.321547    306959
weighted avg   0.800857  0.862311  0.803439    306959
```

Out[37]:

Predicted	2	3	All
Actual			
1	189	3922	4111
2	767	37384	38151
3	770	263927	264697
All	1726	305233	306959

Figure 20: Random Forest with hyperparameters

10.3 Logistics Regression with & without Hyperparameter

For running a logistic regression model the library required is "from sklearn.linear_model import LogisticRegression" as shown in Figure 3. Figure 21 shows a logistic regression model built with default parameters and the classification report which includes the accuracy, precision, recall and f1 score and the confusion matrix. Figures 22 shows a logistics regression model built with hyperparameters.

10.4 Decision Tree with & without Hyperparameter

For running a decision tree model the library required are "from sklearn.tree import DecisionTreeClassifier" as shown in Figure 3. Figure 23 shows a random forest model built with default parameters and the classification report which includes the accuracy, precision, recall and f1 score and the confusion matrix. Figures 24 show a decision tree model built with hyperparameters.

10.5 Accuracy of all Models

Figure 25 shows the accuracy of all the models.

Logistic Regression ¶

In [27]:
```
lr = LogisticRegression()
# Fit the model on the trainng data.
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
sk_report = classification_report(
    digits=6,
    y_true=y_test,
    y_pred=y_pred)
print("Accuracy", round(accuracy_score(y_pred, y_test)*100,2))
print(sk_report)
pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'], margins=True)
```

```
Accuracy 86.23
              precision    recall  f1-score   support

           1   0.000000  0.000000  0.000000      4111
           2   0.000000  0.000000  0.000000     38151
           3   0.862323  0.999928  0.926042    264697

    accuracy                       0.862258    306959
   macro avg   0.287441  0.333309  0.308681    306959
weighted avg   0.743599  0.862258  0.798545    306959
```

Out[27]:

Predicted	1	3	All
Actual			
1	0	4111	4111
2	4	38147	38151
3	19	264678	264697
All	23	306936	306959

Figure 21: Logistics Regression without Hyperparameters

Logistic Regression with Hyperparameter tuning

In [29]:
```
from sklearn.linear_model import LogisticRegressionCV
lr = LogisticRegressionCV(cv=3, random_state=0,multi_class='multinomial')
# Fit the model on the traing data.
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
sk_report = classification_report(digits=6,y_true=y_test,y_pred=y_pred)
print("Accuracy", round(accuracy_score(y_pred, y_test)*100,2))
print(sk_report)
pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'], margins=True)
```

```
Accuracy 86.23
              precision    recall  f1-score   support

           1   0.000000  0.000000  0.000000      4111
           2   0.000000  0.000000  0.000000     38151
           3   0.862319  0.999989  0.926065    264697

    accuracy                       0.862311    306959
   macro avg   0.287440  0.333330  0.308688    306959
weighted avg   0.743595  0.862311  0.798565    306959
```

Out[29]:

Predicted	1	3	All
Actual			
1	0	4111	4111
2	0	38151	38151
3	3	264694	264697
All	3	306956	306959

Figure 22: Logistics Regression with Hyperparameters

Decision Tree

In [28]:
```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
Y_pred = decision_tree.predict(X_test)
acc_decision_tree1 = round(decision_tree.score(X_test, y_test) * 100, 2)
sk_report = classification_report(digits=6,y_true=y_test,y_pred=Y_pred)
print("Accuracy", acc_decision_tree1)
print(sk_report)
### Confusion Matrix
pd.crosstab(y_test, Y_pred, rownames=['Actual'], colnames=['Predicted'], margins=True)
```

```
Accuracy 75.36
              precision    recall  f1-score   support

           1   0.034483  0.042569  0.038101      4111
           2   0.160482  0.188750  0.173472     38151
           3   0.871364  0.846069  0.858531    264697

    accuracy                       0.753612    306959
   macro avg   0.355443  0.359129  0.356701    306959
weighted avg   0.771803  0.753612  0.762399    306959
```

Out[28]:

Predicted	1	2	3	All
Actual				
1	175	907	3029	4111
2	918	7201	30032	38151
3	3982	36763	223952	264697
All	5075	44871	257013	306959

Figure 23: Decision Tree without Hyperparameters

Decision Tree hyperparameters tuning

All we are going to do is find the best values for mininum sample leaf and maximum features to get the best score.

In [30]:
```
decision_tree = DecisionTreeClassifier(min_samples_leaf=12, max_features=4)
decision_tree.fit(X_train, y_train)
Y_pred = decision_tree.predict(X_test)
acc_decision_tree1 = round(decision_tree.score(X_test, y_test) * 100, 2)
sk_report = classification_report(digits=6, y_true=y_test,y_pred=Y_pred)
print("Accuracy", acc_decision_tree1)
print(sk_report)
### Confusion Matrix
pd.crosstab(y_test, Y_pred, rownames=['Actual'], colnames=['Predicted'], margins=True)
```

```
Accuracy 85.69
              precision    recall  f1-score   support

           1   0.153846  0.000973  0.001934      4111
           2   0.316212  0.044376  0.077830     38151
           3   0.866592  0.987340  0.923034    264697

    accuracy                       0.856932    306959
   macro avg   0.445550  0.344230  0.334266    306959
weighted avg   0.788642  0.856932  0.805650    306959
```

Out[30]:

Predicted	1	2	3	All
Actual				
1	4	329	3778	4111
2	3	1693	36455	38151
3	19	3332	261346	264697
All	26	5354	301579	306959

Figure 24: Decision Tree with Hyperparameters

Accuracy of all Machine Learning Models

In [41]:
```
results=pd.DataFrame({ "Algorithm":["Random Forest","Logistic Regression","Decision Tree"],
                       "Accuracy":[acc_random_forest1,round(accuracy_score(y_pred, y_test)*100,2), acc_decision_tree1]})

results.sort_values(ascending=False,by="Accuracy")
```

Out[41]:

	Algorithm	Accuracy
0	Random Forest	86.24
1	Logistic Regression	86.23
2	Decision Tree	85.67

Figure 25: Accuracy of all Model