Candidate number : 560966

Big Data For Decision making Individual Coursework

Accident Severity Prediction

All the data preprocessing steps like missing value treatment,outlier treatment and dummy variable creation are all done in the Group part of this problems.

In the individual part I will do the feature scaling , baseline model, and build various model to predict accident severity. The data mainly focus on the accidents that occured during the winter season in UK as accidents tend to be more in winter season

Table of contents

1. Introduction and Business Objective

There is no doubt that the top cause of winter car accidents is ice and snow on the roadways. When the roads are icy and slick, the traction on your tires is less effective. Therefore impacting a huge loss for the Insurance companies.

The more the accidents the higher the claims raised by the insurer, therefore insurance companies are in a stage to introduce new policies from keeping their revenue and profit intact.

Therefore we aim to predict the severity of accident within the United Kingdom during the snow season and suggest "Forever Live" Insurance Company with preplanned policies that take winter prone accidents into consideration. We are going to use the Machine Learning Methods to solve this classification problem keeping Accident Severity as our Target variable

In [440…

```
```

Out[440]:

2. Importing all the necessary libraries

In [ ]:

In [153…
```
import logging
logging.basicConfig()
logging.getLogger("SKLEARNEX").setLevel(logging.ERROR)

import time
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

In [154…
```
from sklearn.metrics import precision_recall_fscore_support, classification_report
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.model_selection import cross_val_predict
```

In [155…
```
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.dummy import DummyClassifier
```

In [ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier
```

In [ ]:
```
from sklearn.metrics import precision_recall_fscore_support
```

In [270…
```
import os
from joblib import dump
```

In [364…
```
from sklearn.tree import  plot_tree
import warnings
warnings.filterwarnings('ignore')
```

3. Loading Data

In [252…
```
Trainset=pd.read_csv('trainingset.csv')
Testset=pd.read_csv('testingset.csv')
```

In [253…
```
Trainset['accident_severity']=Trainset['accident_severity'].astype(str)
Testset['accident_severity']=Testset['accident_severity'].astype(str)
```

In [ ]:
```
```

We need to drop the categorical variables after creating dummy variables for each of them

In [254…
```
Trainset.drop(['road_type','light_conditions','weather_conditions','road_surface_c
```

In [255…
```
Testset.drop(['road_type','light_conditions','weather_conditions','road_surface_co
```

In [227…
```
print(Trainset.shape)
print(Testset.shape)
```

```
(8159, 45)
(2027, 45)
```

Creating seperate dataframes for Trarget and predictors for both Training and Testing set

In [256…
```
Xtrain=Trainset.drop('accident_severity',axis=1)
```

In [257…
```
Ytrain=Trainset['accident_severity'].copy()
```

In [258…
```
Xtest=Testset.drop('accident_severity',axis=1)
Ytest=Testset['accident_severity'].copy()
```

In [231…
```
print(Ytrain.shape)
print(Xtrain.shape)
```

```
(8159,)
(8159, 44)
```

In [232…
```
print(Ytest.shape)
print(Xtest.shape)
```

```
(2027,)
(2027, 44)
```

Trainset have 8189 rows and 44 columns and testset have 2027 columns and 44 columns

In [233…
```
Xtrain.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8159 entries, 0 to 8158
Data columns (total 44 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   number_of_casualties         8159 non-null   float64
 1   age_of_driver                8159 non-null   float64
 2   engine_capacity_cc           8159 non-null   float64
 3   age_of_vehicle               8159 non-null   float64
 4   casualty_severity            8159 non-null   int64  
 5   HourOfDay                    8159 non-null   float64
 6   road_type_1                  8159 non-null   int64  
 7   road_type_2                  8159 non-null   int64  
 8   road_type_3                  8159 non-null   int64  
 9   road_type_6                  8159 non-null   int64  
 10  road_type_7                  8159 non-null   int64  
 11  road_type_9                  8159 non-null   int64  
 12  light_conditions_1           8159 non-null   int64  
 13  light_conditions_4           8159 non-null   int64  
 14  light_conditions_5           8159 non-null   int64  
 15  light_conditions_6           8159 non-null   int64  
 16  light_conditions_7           8159 non-null   int64  
 17  weather_conditions_1         8159 non-null   int64  
 18  weather_conditions_2         8159 non-null   int64  
 19  weather_conditions_3         8159 non-null   int64  
 20  weather_conditions_4         8159 non-null   int64  
 21  weather_conditions_5         8159 non-null   int64  
 22  weather_conditions_6         8159 non-null   int64  
 23  weather_conditions_7         8159 non-null   int64  
 24  weather_conditions_8         8159 non-null   int64  
 25  weather_conditions_9         8159 non-null   int64  
 26  road_surface_conditions_1    8159 non-null   int64  
 27  road_surface_conditions_2    8159 non-null   int64  
 28  road_surface_conditions_3    8159 non-null   int64  
 29  road_surface_conditions_4    8159 non-null   int64  
 30  road_surface_conditions_5    8159 non-null   int64  
 31  road_surface_conditions_9    8159 non-null   int64  
 32  sex_of_driver_1              8159 non-null   int64  
 33  sex_of_driver_2              8159 non-null   int64  
 34  sex_of_driver_3              8159 non-null   int64  
 35  Month_December               8159 non-null   int64  
 36  Month_January                8159 non-null   int64  
 37  Day_Friday                   8159 non-null   int64  
 38  Day_Monday                   8159 non-null   int64  
 39  Day_Saturday                 8159 non-null   int64  
 40  Day_Sunday                   8159 non-null   int64  
 41  Day_Thursday                 8159 non-null   int64  
 42  Day_Tuesday                  8159 non-null   int64  
 43  Day_Wednesday                8159 non-null   int64  
dtypes: float64(5), int64(39)
memory usage: 2.7 MB
```

4. Feature scaling

As the scales of my predictors were different as seen from the distributions of variables in the group assignment part, I am going to scale my variables which may improve my model. I am doing the same thing for testing set as well.

Target variable is not changed Standard scaled in used to do scaling

In [259…

```
scaler = StandardScaler()



# fit_transform returns a NumPy array, so we need to put it back
# into a Pandas dataframe
scaled_vals = scaler.fit_transform(Xtrain)
Xtrain = pd.DataFrame(scaled_vals, columns=Xtrain.columns)





# inspect the data
Xtrain.head()
```

Out[259]:

	number_of_casualties	age_of_driver	engine_capacity_cc	age_of_vehicle	casualty_severity	Hour(
0	0.847828	-0.661229	-2.224426	-1.368147	-3.935540	1.0
1	2.284170	-1.271739	-0.548328	0.658004	0.480644	0.6
2	3.002341	0.010334	-0.024641	-1.368147	-1.727448	1.2
3	-0.588515	-0.661229	0.589078	1.578982	0.480644	-0.5
4	-0.588515	-0.844382	0.575573	-0.815560	0.480644	-0.1

5 rows × 44 columns

In [260…

```
scaled_vals_1 = scaler.transform(Xtest)
Xtest = pd.DataFrame(scaled_vals_1, columns=Xtest.columns)
Xtest.head()
```

Out[260]:

	number_of_casualties	age_of_driver	engine_capacity_cc	age_of_vehicle	casualty_severity	Hour(
0	-0.588515	-1.393842	-0.614352	0.473809	0.480644	-2.1
1	-0.588515	-1.027535	1.019732	-0.631365	0.480644	-1.1
2	3.720513	-0.661229	0.586077	-0.815560	-1.727448	1.2
3	-0.588515	0.681896	2.079110	-1.368147	0.480644	-0.9
4	6.593197	-1.149637	0.433022	1.763177	-1.727448	0.0

5 rows × 44 columns

5. Building Models

As our problem is a classification problem where target variable is categorical, we are going to build the following models

1. Random forest

2. Decision Tree

3. K Nearest Neighbour

4. Logistic Regression

5. Gradient Booster

Baseline

We will use mode as the baseline for our model

In [236…

```
Ytrain.value_counts()
```

Out[236]:

```
3    6100
2    1837
1     222
Name: accident_severity, dtype: int64
```

In [237…

```
Ytrain.shape
```

Out[237]:

```
(8159,)
```

In [261…

```
dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(Xtrain, Ytrain)
yhat_train = dummy_clf.predict(Xtrain)

evaluate_model(dummy_clf, Ytrain, Xtrain)
```

```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00       222
           2       0.00      0.00      0.00      1837
           3       0.75      1.00      0.86      6100

    accuracy                           0.75      8159
   macro avg       0.25      0.33      0.29      8159
weighted avg       0.56      0.75      0.64      8159
```

We get F score of 0.29 for the baseline model

Random forest

In [262…

```
%%time


rf = RandomForestClassifier(random_state=7)


param_grid = {
    'n_estimators': [10,  200, 500],
    'max_depth': [5, 7, 15],
    'min_samples_split': [5, 10]
}


grid_search = GridSearchCV(rf, param_grid, cv=5,
                           scoring='f1_macro',
                           return_train_score=True)
grid_search.fit(Xtrain, Ytrain)
```

```
Wall time: 4min 39s
```

Out[262]:

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
  ▸ RandomForestClassifier
```

In [263…

```
grid_search.best_estimator_
```

Out[263]:

```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=15, min_samples_split=5, n_estimators=50
0,
                       random_state=7)
```

After building the model we get the best hyperparameters to be

max_depth=15

min_samples_split=5

n_estimators = 500

random_state = 7

In [264…

```
grid_search.best_score_
```

Out[264]:

```
0.8626284605104113
```

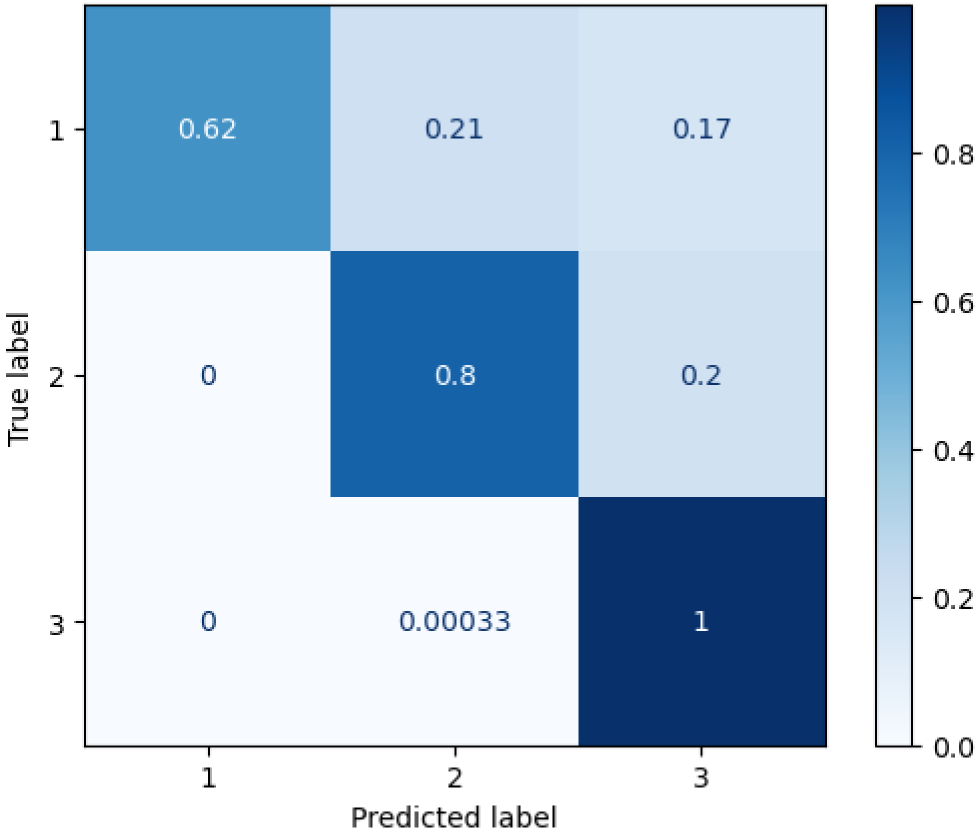
F score for this model is 86.26%

In [242…

```
yhat = cross_val_predict(grid_search.best_estimator_, Xtrain, Ytrain, cv=10)
ConfusionMatrixDisplay.from_predictions(Ytrain, yhat,
                                        labels=grid_search.best_estimator_.classes_
                                        normalize="true",
                                        cmap=plt.cm.Blues)
```

Out[242]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1fc81668430>
```



The model predicts most predictions accurately

Record the result of the best model

In [267…

```
best_model = grid_search.cv_results_["rank_test_score"].tolist().index(1)
best_model
```

Out[267]: 14

In [268…

```
# Keep the feature importance in to a separate variable
Feature_importances = grid_search.best_estimator_.feature_importances_


for k, v in sorted(zip(Feature_importances, Xtrain.columns), reverse=True):
    print(f"{v}: {k}")
```

```
casualty_severity: 0.7073529047040515
number_of_casualties: 0.06016218356512073
engine_capacity_cc: 0.0314773649125223
HourOfDay: 0.03085611335496713
age_of_driver: 0.030526145578245397
age_of_vehicle: 0.0248675676816525
light_conditions_6: 0.00868262210672561
road_type_6: 0.006397053510131872
Day_Tuesday: 0.0049034801841173785
road_surface_conditions_2: 0.004794589356810566
Month_January: 0.004748299536160826
Month_December: 0.004730082118833791
road_surface_conditions_1: 0.004592996826600017
weather_conditions_1: 0.004527379354848166
Day_Friday: 0.004322110610173159
Day_Monday: 0.0041990138706214045
light_conditions_1: 0.0041924166402182725
light_conditions_4: 0.0041878452630675519
Day_Sunday: 0.0040477465020269275
road_type_3: 0.0040095498038726085
Day_Saturday: 0.003994955881912801
Day_Wednesday: 0.003936490948805728
Day_Thursday: 0.003782103323857343
weather_conditions_2: 0.0037244222339690215
sex_of_driver_1: 0.003625041114413088
sex_of_driver_2: 0.003260768231613393
weather_conditions_5: 0.002746285300102328
road_surface_conditions_4: 0.0023449893889475057
weather_conditions_8: 0.0020316595394103834
weather_conditions_7: 0.001979921449324208
weather_conditions_3: 0.001879464068729974
weather_conditions_9: 0.0016691203881523478
road_type_1: 0.0016533082555728234
weather_conditions_4: 0.0015661070181310758
road_surface_conditions_3: 0.001454343711585736
light_conditions_7: 0.0012410281768249682
road_type_7: 0.0011197224971786347
road_type_2: 0.0009184348829415532
light_conditions_5: 0.0007969584375724788
sex_of_driver_3: 0.0007132854024502692
road_type_9: 0.0006468492761668132
weather_conditions_6: 0.0006108690156724233
road_surface_conditions_5: 0.0005677306263809966
road_surface_conditions_9: 0.0001586753495164607
```

Casuality severity is the most important predictor in my model with 70% importance followed by number of casuality and engine capacity

We save model in to a disc for future reference

In [271…

```python
if not os.path.exists("models"):
    os.makedirs("models")

dump(grid_search.best_estimator_, 'models/rf-clf.joblib')
```

Out[271]:

```
['models/rf-clf.joblib']
```

Decision Tree

In [278…

```python
%%time
```

```
# initialize decision tree model
deci_tree = DecisionTreeClassifier(random_state=7)


param_grid = {
    'max_depth': [5, 7, 15],
    'min_samples_split': [5, 10],

}


grid_search_dt = GridSearchCV(dt, param_grid, cv=5,
                           scoring='f1_macro',
                           return_train_score=True)
grid_search_dt.fit(Xtrain, Ytrain)
```

Wall time: 3.96 s

Out[278]:

```
GridSearchCV
estimator: DecisionTreeClassifier
DecisionTreeClassifier
```

In [282…

```
grid_search_dt.best_estimator_
```

Out[282]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=7, min_samples_split=5, random_state=7)
```

After building the model we get the best hyperparameters to be

max_depth=7

min_samples_split=5

random_state = 7

In [284…

```
grid_search_dt.best_score_
```

Out[284]:

```
0.8698867044240807
```

F score for this model is 86.9%

In [286…

```
bmodel=DecisionTreeClassifier(max_depth= 7, min_samples_split=5,random_state=7).fi
```

We save the best fit model

In [287…

```
text_=tree.export_text(bmodel)
print(text_)
```

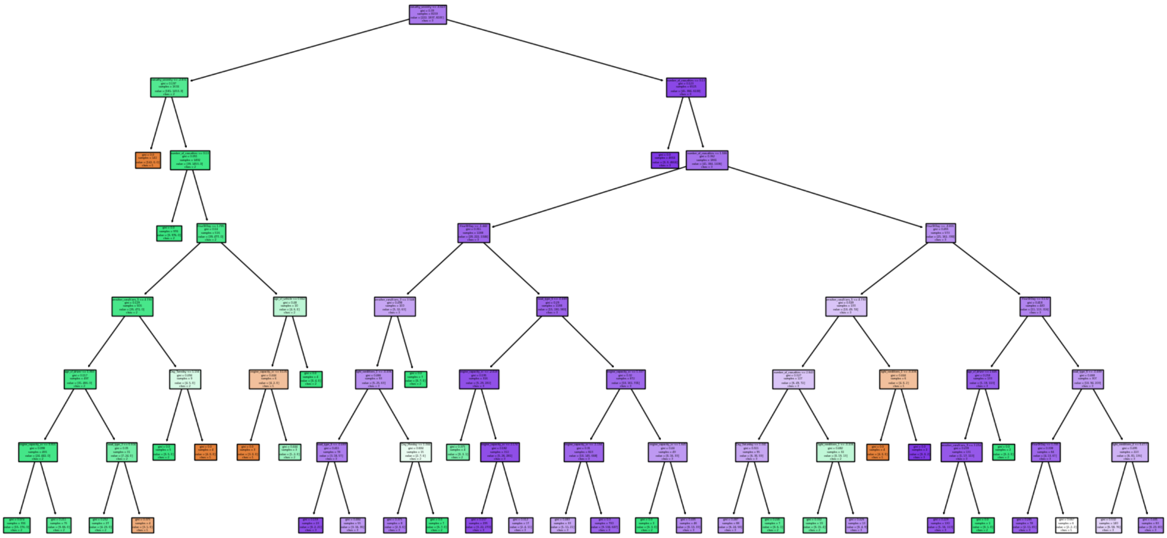
```
|--- feature_4 <= -0.62
|   |--- feature_4 <= -2.83
|   |   |--- class: 1
|   |--- feature_4 >  -2.83
|   |   |--- feature_0 <= 0.13
|   |   |   |--- class: 2
|   |   |--- feature_0 >  0.13
|   |   |   |--- feature_5 <= 1.71
|   |   |   |   |--- feature_21 <= 4.79
|   |   |   |   |   |--- feature_1 <= 1.90
|   |   |   |   |   |   |--- feature_2 <= 0.92
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- feature_2 >  0.92
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_1 >  1.90
|   |   |   |   |   |   |--- feature_8 <= 0.90
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- feature_8 >  0.90
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- feature_21 >  4.79
|   |   |   |   |   |--- feature_42 <= 1.13
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_42 >  1.13
|   |   |   |   |   |   |--- class: 1
|   |   |   |--- feature_5 >  1.71
|   |   |   |   |--- feature_3 <= 0.84
|   |   |   |   |   |--- feature_2 <= 0.14
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_2 >  0.14
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |--- feature_3 >  0.84
|   |   |   |   |   |--- class: 2
|--- feature_4 >  -0.62
|   |--- feature_0 <= 0.13
|   |   |--- class: 3
|   |--- feature_0 >  0.13
|   |   |--- feature_0 <= 1.57
|   |   |   |--- feature_5 <= -1.44
|   |   |   |   |--- feature_19 <= 3.55
|   |   |   |   |   |--- feature_12 <= -0.21
|   |   |   |   |   |   |--- feature_9 <= -0.49
|   |   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |   |--- feature_9 >  -0.49
|   |   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- feature_12 >  -0.21
|   |   |   |   |   |   |--- feature_38 <= 1.04
|   |   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |   |--- feature_38 >  1.04
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |--- feature_19 >  3.55
|   |   |   |   |   |--- class: 2
|   |   |   |--- feature_5 >  -1.44
|   |   |   |   |--- feature_9 <= -0.49
|   |   |   |   |   |--- feature_2 <= -2.02
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_2 >  -2.02
|   |   |   |   |   |   |--- feature_2 <= 2.17
|   |   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |   |--- feature_2 >  2.17
|   |   |   |   |   |   |   |--- class: 3
|   |   |   |   |--- feature_9 >  -0.49
|   |   |   |   |   |--- feature_2 <= 1.34
|   |   |   |   |   |   |--- feature_2 <= -1.13
|   |   |   |   |   |   |   |--- class: 3
```

```
|   |   |   |   |   |   |--- feature_2 >  -1.13
|   |   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |--- feature_2 >  1.34
|   |   |   |   |   |   |--- feature_2 <= 1.64
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- feature_2 >  1.64
|   |   |   |   |   |   |   |--- class: 3
|   |   |--- feature_0 >  1.57
|   |   |   |--- feature_5 <= -0.66
|   |   |   |   |--- feature_21 <= 4.79
|   |   |   |   |   |--- feature_0 <= 2.64
|   |   |   |   |   |   |--- feature_39 <= 1.04
|   |   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |   |--- feature_39 >  1.04
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_0 >  2.64
|   |   |   |   |   |   |--- feature_12 <= -0.21
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |   |--- feature_12 >  -0.21
|   |   |   |   |   |   |   |--- class: 3
|   |   |   |   |--- feature_21 >  4.79
|   |   |   |   |   |--- feature_12 <= -0.21
|   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_12 >  -0.21
|   |   |   |   |   |   |--- class: 3
|   |   |   |--- feature_5 >  -0.66
|   |   |   |   |--- feature_5 <= 0.13
|   |   |   |   |   |--- feature_1 <= 2.61
|   |   |   |   |   |   |--- feature_25 <= 3.25
|   |   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |   |--- feature_25 >  3.25
|   |   |   |   |   |   |   |--- class: 2
|   |   |   |   |   |--- feature_1 >  2.61
|   |   |   |   |   |   |--- class: 2
|   |   |   |   |--- feature_5 >  0.13
|   |   |   |   |   |--- feature_9 <= -0.49
|   |   |   |   |   |   |--- feature_5 <= 1.71
|   |   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |   |--- feature_5 >  1.71
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- feature_9 >  -0.49
|   |   |   |   |   |   |--- feature_13 <= 0.47
|   |   |   |   |   |   |   |--- class: 3
|   |   |   |   |   |   |--- feature_13 >  0.47
|   |   |   |   |   |   |   |--- class: 3
```

We plot the decision tree with the best hyperparameters

In [352…

```
plt.figure(figsize=(20,10))
plot_tree(bmodel, feature_names=Xtrain.columns, class_names=['1', '2','3'], filled
plt.show()
```

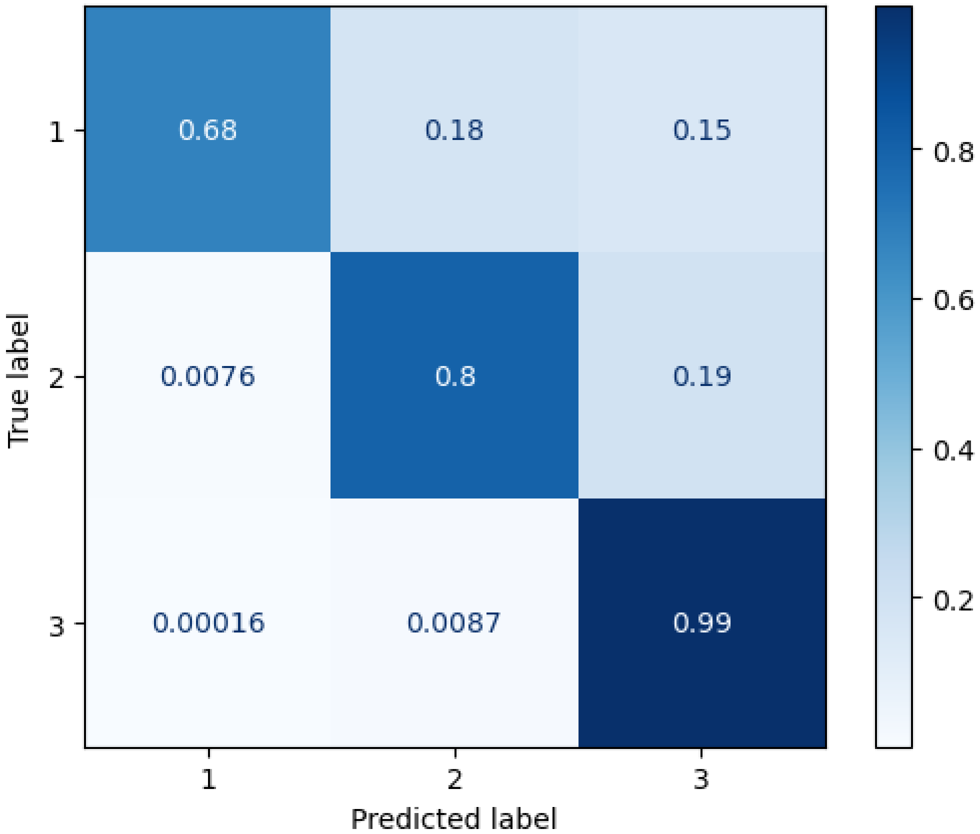


f score of the model is 86.98%

In [295…

```
yhat_dt = cross_val_predict(grid_search_dt.best_estimator_, Xtrain, Ytrain, cv=10)
ConfusionMatrixDisplay.from_predictions(Ytrain, yhat_dt,
                                        labels=grid_search_dt.best_estimator_.clas
                                        normalize="true",
                                        cmap=plt.cm.Blues)
```

Out[295]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1fc8197c850>



The model predicts most predictions with a certain level of accuracy

Record the result of the best model

In [296…

```
best_model_dt = grid_search_dt.cv_results_["rank_test_score"].tolist().index(1)
best_model_dt
```

Out[296]:
```
2
```

In [298…
```
# Keep the feature importance in to a separate variable
Feature_importances_dt = grid_search_dt.best_estimator_.feature_importances_


for k, v in sorted(zip(Feature_importances_dt, Xtrain.columns), reverse=True):
    print(f"{v}: {k}")
```

```
casualty_severity: 0.9058504052117596
number_of_casualties: 0.05719329413298721
HourOfDay: 0.008019132723681932
engine_capacity_cc: 0.005548526617680046
road_type_6: 0.004112413771174869
light_conditions_1: 0.003319140369600976
weather_conditions_3: 0.002551069489222565
Day_Monday: 0.0023858923280498786
weather_conditions_5: 0.0023378960196935835
age_of_driver: 0.0018578064055881506
Day_Tuesday: 0.0017479064674358088
Day_Saturday: 0.0015624169126841022
light_conditions_4: 0.0010874876587518563
road_type_3: 0.0009925947345317597
age_of_vehicle: 0.0008389951043691881
weather_conditions_9: 0.0005950220527885495
weather_conditions_8: 0.0
weather_conditions_7: 0.0
weather_conditions_6: 0.0
weather_conditions_4: 0.0
weather_conditions_2: 0.0
weather_conditions_1: 0.0
sex_of_driver_3: 0.0
sex_of_driver_2: 0.0
sex_of_driver_1: 0.0
road_type_9: 0.0
road_type_7: 0.0
road_type_2: 0.0
road_type_1: 0.0
road_surface_conditions_9: 0.0
road_surface_conditions_5: 0.0
road_surface_conditions_4: 0.0
road_surface_conditions_3: 0.0
road_surface_conditions_2: 0.0
road_surface_conditions_1: 0.0
light_conditions_7: 0.0
light_conditions_6: 0.0
light_conditions_5: 0.0
Month_January: 0.0
Month_December: 0.0
Day_Wednesday: 0.0
Day_Thursday: 0.0
Day_Sunday: 0.0
Day_Friday: 0.0
```

Casuality severity is the most important predictor in my decision tree model as well

Save the model in to a disc for future reference

In [297…
```
if not os.path.exists("models"):
    os.makedirs("models")
```

```
dump(grid_search_dt.best_estimator_, 'models/dt-clf.joblib')
```

Out[297]:
```
['models/dt-clf.joblib']
```

K nearest neighbour

In [331…

```
%%time



knn = KNeighborsClassifier()

# specify the hyperparameters and their values
param_grid = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance'],
    'p': [1,2]
}


grid_search_knn = GridSearchCV(knn, param_grid, cv=5,
                              scoring='f1_macro',
                              return_train_score=True)
grid_search_knn.fit(Xtrain, Ytrain)
```

```
Wall time: 1min 44s
```

Out[331]:
```
GridSearchCV
estimator: KNeighborsClassifier
KNeighborsClassifier
```

In [337…

```
grid_search_knn.best_estimator_
```

Out[337]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3, p=1, weights='distance')
```

After building the model we get the best hyperparameters to be

n_neighbors=3

p=1

weights='distance'

In [338…

```
grid_search_knn.best_score_
```

Out[338]:
```
0.7591654579192466
```

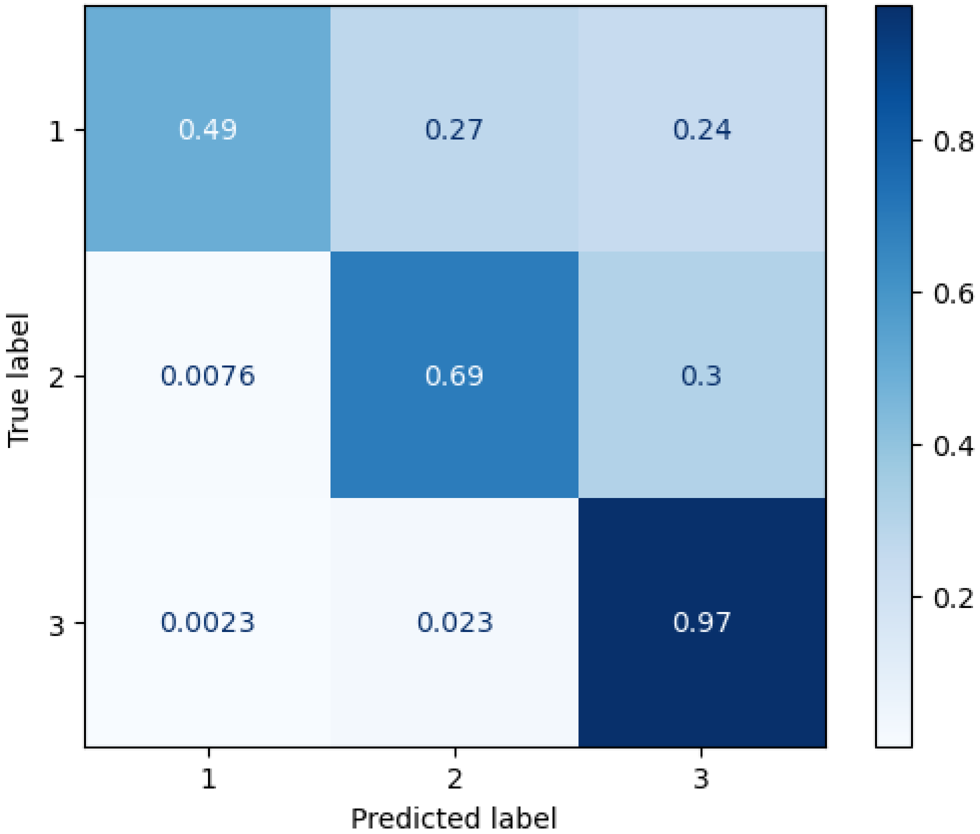
Score for the best model is 75.91 % which is less than the previous two models

In [340…

```
yhat_knn = cross_val_predict(grid_search_knn.best_estimator_, Xtrain, Ytrain, cv=1(
ConfusionMatrixDisplay.from_predictions(Ytrain, yhat_knn,
```

```
                                          labels=grid_search_knn.best_estimator_.cla
                                          normalize="true",
                                          cmap=plt.cm.Blues)
```

Out[340]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1fc8cc3a3a0>



From the confusion matrix we can see that the model performs poorely in performing class 1

In [341…
```
bknn=KNeighborsClassifier(n_neighbors=3, p=1, weights='distance').fit(Xtrain,Ytrai
```

Best fit model is saved in to a variable

In [344…
```
best_model_knn = grid_search_knn.cv_results_["rank_test_score"].tolist().index(1)
best_model_knn
```

Out[344]: 1

save the model in to a disc for future reference

In [350…
```
if not os.path.exists("models"):
    os.makedirs("models")

dump(grid_search_knn.best_estimator_, 'models/knn-clf.joblib')
```

Out[350]: ['models/knn-clf.joblib']

In [ ]:

Logistic Regression

In [365…
```
%%time
```

```
logreg = LogisticRegression(random_state=7)


param_grid = {
    'penalty': ['l1', 'l2', 'elasticnet', 'none'],
    'C': [0.001, 0.01, 0.1, 1, 10, 100],
    'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
}


grid_search_logreg = GridSearchCV(logreg, param_grid, cv=5,
                                  scoring='f1_macro',
                                  return_train_score=True)
grid_search_logreg.fit(Xtrain, Ytrain)
```

```
Wall time: 6min 50s
```

Out[365]:

```
GridSearchCV
estimator: LogisticRegression
LogisticRegression
```

In [368…

```
grid_search_logreg.best_estimator_
```

Out[368]:

```
LogisticRegression
LogisticRegression(C=0.1, penalty='l1', random_state=7, solver='saga')
```

After building the model we get the best hyperparameters to be

C=0.1

penalty = 'l1'

solver = 'saga'

In [369…

```
grid_search_logreg.best_score_
```

Out[369]:

```
0.8485469563611219
```

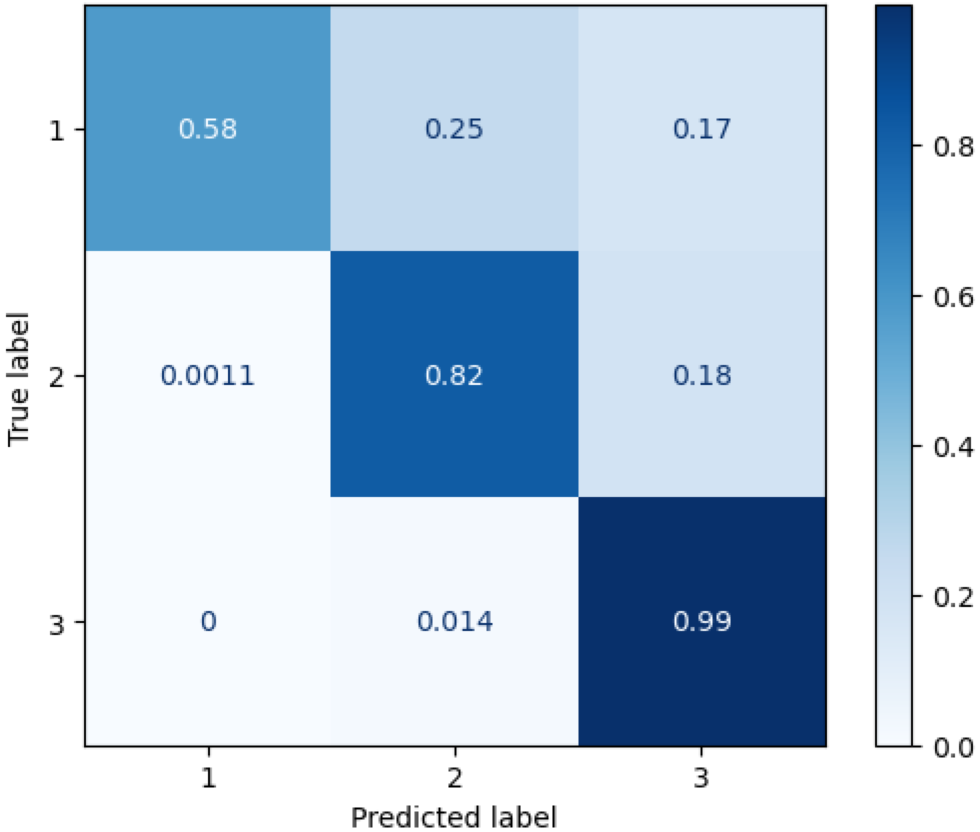
score for the best model is 84.85%

In [370…

```
yhat_logi = cross_val_predict(grid_search_logreg.best_estimator_, Xtrain, Ytrain, 
ConfusionMatrixDisplay.from_predictions(Ytrain, yhat_logi,
                                        labels=grid_search_logreg.best_estimator_.
                                        normalize="true",
                                        cmap=plt.cm.Blues)
```

Out[370]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1fc9be3dcd0>
```



Model predicts accurately for class 2 and 3 but performs poorly for class 1 that is fatal

In [371…

```
blogi=LogisticRegression(C=0.1, penalty='l1', random_state=7, solver='saga').fit(X
```

I saved the best fit model in to a variable

In [374…

```
best_model_logreg = grid_search_logreg.cv_results_["rank_test_score"].tolist().ind
best_model_logreg
```

Out[374]: 44

In [ ]:

saved the model in to a disc

In [375…

```
if not os.path.exists("models"):
    os.makedirs("models")

dump(grid_search_logreg.best_estimator_, 'models/logi-clf.joblib')
```

Out[375]: ['models/logi-clf.joblib']

Gradient booster

In [429…

```
%%time


# initialize Gradient Boosting model
gb = GradientBoostingClassifier()
```

```
param_grid = {
    'n_estimators': [50, 100, 150],
    'learning_rate': [0.01, 0.1, 1.0],
    'max_depth': [3, 5, 7],
    'min_samples_split': [2, 4, 6],
    'min_samples_leaf': [1, 2, 4]
}


grid_search_gb = GridSearchCV(gb, param_grid, cv=5,
                              scoring='f1_macro',
                              return_train_score=True)
grid_search_gb.fit(Xtrain, Ytrain)
```

```
Wall time: 3h 50min 31s
```

Out[429]:

```
▸ GridSearchCV
▸ estimator: GradientBoostingClassifier
    ▸ GradientBoostingClassifier
```

In [431…

```
grid_search_gb.best_estimator_
```

Out[431]:

```
▾ GradientBoostingClassifier
GradientBoostingClassifier(max_depth=7, min_samples_split=4, n_estimators
=150)
```

In [432…

```
grid_search_logreg.best_score_
```

Out[432]:

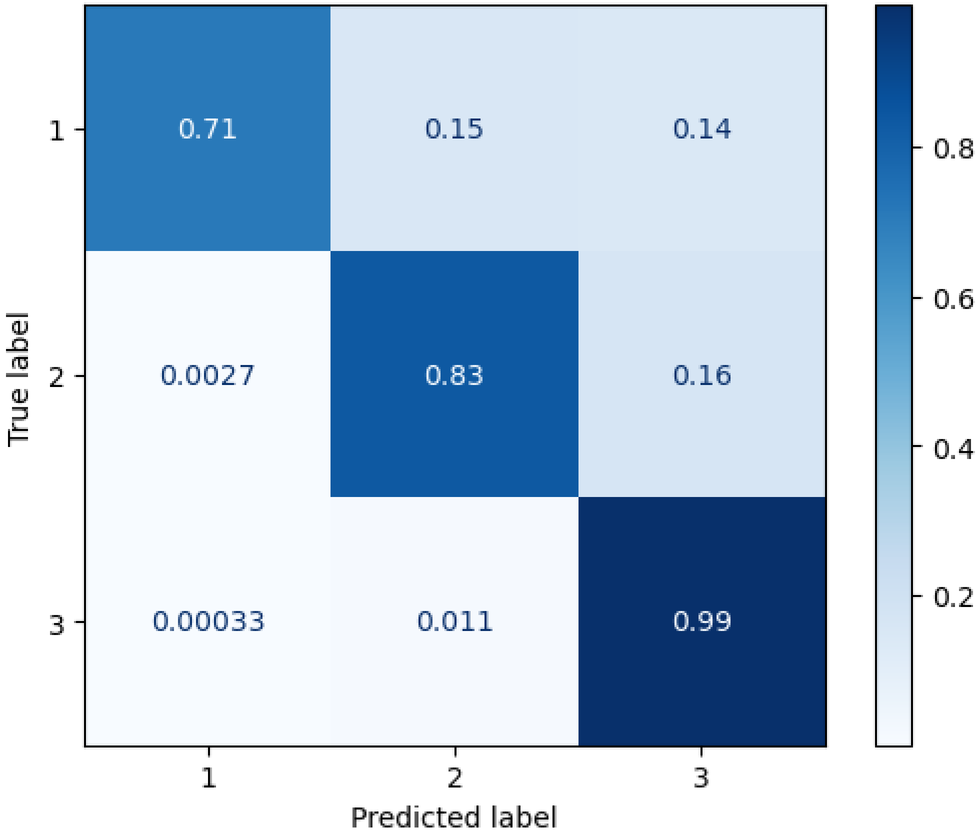
```
0.8485469563611219
```

In [433…

```
yhat_gb = cross_val_predict(grid_search_gb.best_estimator_, Xtrain, Ytrain, cv=10)
ConfusionMatrixDisplay.from_predictions(Ytrain, yhat_gb,
                                        labels=grid_search_gb.best_estimator_.clas
                                        normalize="true",
                                        cmap=plt.cm.Blues)
```

Out[433]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1fca400e640>
```



In [ ]:
```
best_model_logreg = grid_search_logreg.cv_results_["rank_test_score"].tolist().ind
best_model_logreg
```

6. Testing the best models on test data

In [377…
```
best_rf = load("models/rf-clf.joblib")
best_dt = load("models/dt-clf.joblib")
```

Random forest

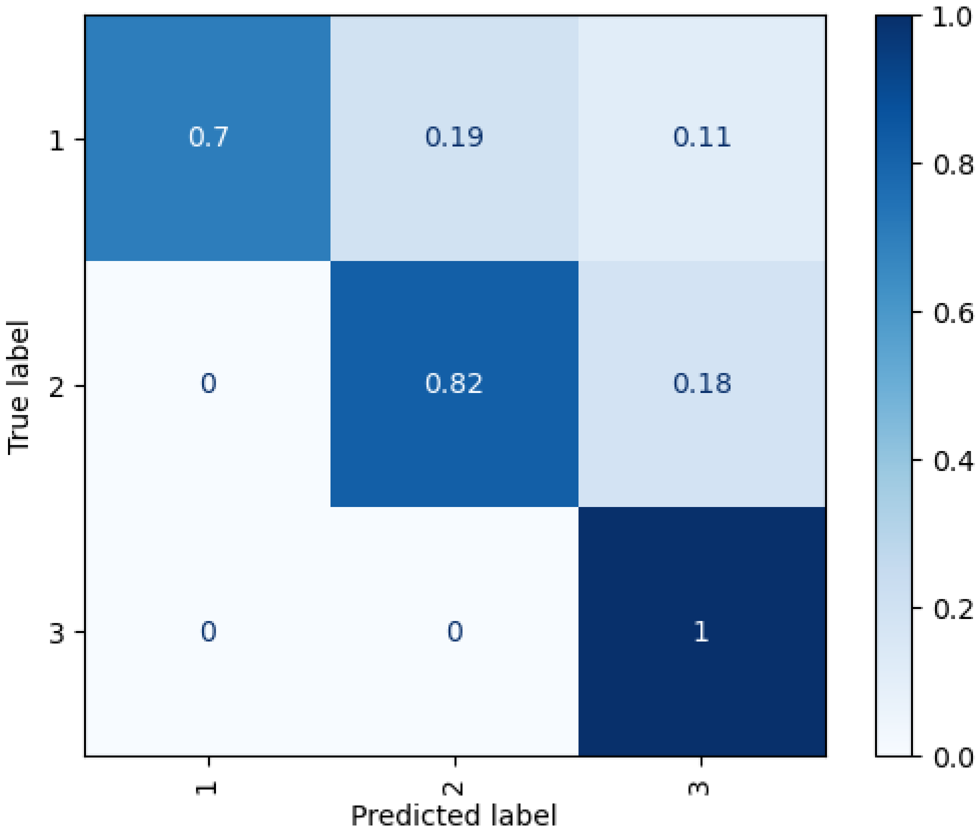
In [379…
```
# rf
yhat_rf= best_rf.predict(Xtest)

# micro-averaged precision, recall and f-score
p, r, f, s = precision_recall_fscore_support(Ytest, yhat_rf, average="macro")
print("Random Forest:")
print(f"Precision: {p}")
print(f"Recall: {r}")
print(f"F score: {f}")
```

```
Random Forest:
Precision: 0.9732932428967671
Recall: 0.8406678119906612
F score: 0.8956932791508293
```

In [380…
```
ConfusionMatrixDisplay.from_predictions(Ytest, yhat_rf, labels=best_rf.classes_,
                                        xticks_rotation="vertical", normalize="tru
                                        cmap=plt.cm.Blues)
```

Out[380]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1fc9e227940>
```



The model accurately predict class 3 (slight) and class 2 (severe) but its prediction is a little low for class 1 (fatal)

Decision tree

In [416…

```
# dt
yhat_dt= best_dt.predict(Xtest)

# micro-averaged precision, recall and f-score
p, r, f, s = precision_recall_fscore_support(Ytest, yhat_dt, average="macro")
print("Random Forest:")
print(f"Precision: {p}")
print(f"Recall: {r}")
print(f"F score: {f}")
```

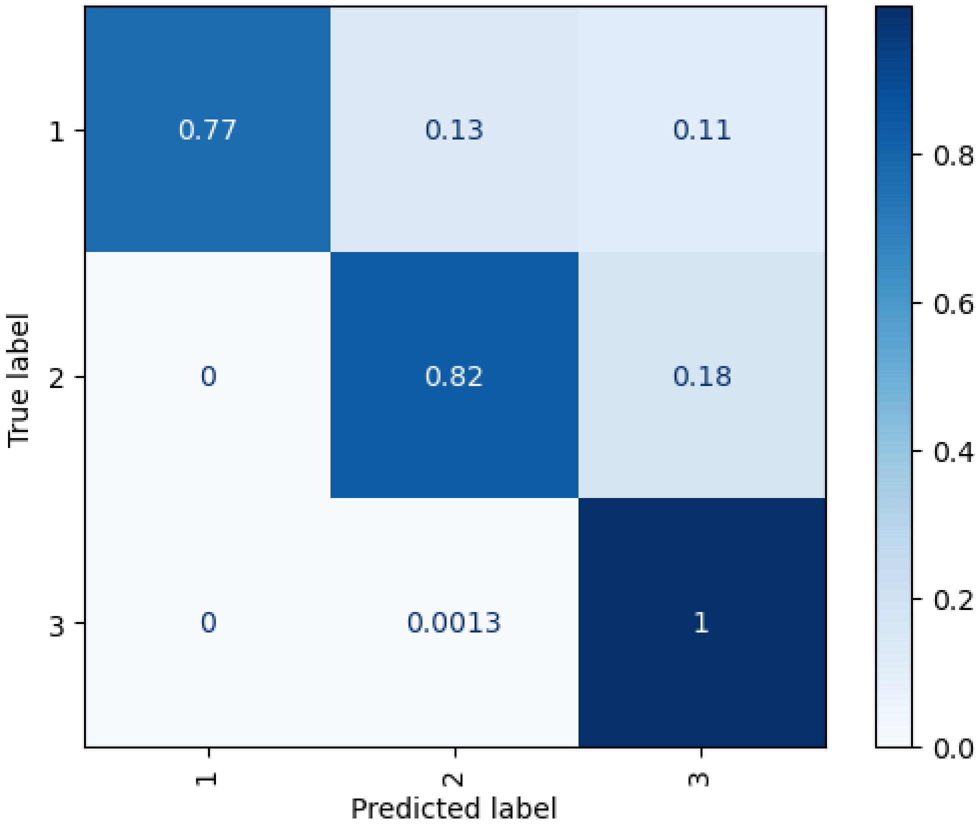
```
Random Forest:
Precision: 0.9745044107058684
Recall: 0.8628793348704957
F score: 0.9110024032466181
```

In [417…

```
ConfusionMatrixDisplay.from_predictions(Ytest, yhat_dt, labels=best_dt.classes_,
                                        xticks_rotation="vertical", normalize="tru
                                        cmap=plt.cm.Blues)
```

Out[417]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1fc9f3176a0>
```



Similar to random forest ,the model accurately predict class 3 (slight) and class 2 (severe) but its prediction is a little low for class 1 (fatal)

it also has similar result for training

7. Evaluation of results

It can be seen from results of above models that both random forest and decision tree produce good accuracy score for both train and test set which means that the model succeeds in correctly predicting majority of the prediction. However as there is problem of class imbalance in the data,It is possible that the model may perform inconsistently on the future unknown data efforts were done to deal with this problem using oversampling which resulted in inconsistent accuracy results and hence that plan was dropped.

Out of the five models created, least accurate model was K Nearest Neighbour, but the worst practical model seems to be gradient boost. Eventhough the model has better f score than KNN, it took almost 4 hour to run which makes it a very slow model.

```
In [ ]:
```

8. Possible future improvement and Business scenario

1. The main issue with the data was about the class imbalace and this could be solved using adding more balanced data where minority class is not under represented.

2. Ensemble methods could be used where models will be trained on different subsets of data and could potentially improve the performance on unseen data could possibily result in a more generalised model. Bagging is one way to this.1.
3. Weighting methods could be used to give more weights to undersampled data so that the model focuses more on them

The model could be used in the following scenarios

1. For insurance company to build insuarance plan based on accident severity and casuality severity, as casuality severity in a strong predictor this model may work quite well in that scenario
2. Insurance company could use this model to create special insurance plans for different age groups

The company can link this model to their database and regularly update the model so that it can perform on future data

Model Deployement

Decision Tree was the best model from the results of my work and hence I am Deploying that model

In [447…

```
import pickle
pikle=open('classifier.pkl','wb')
pickle.dump(bmodel,pikle)
pikle.close()
```

End

Reference

Pekar, V. (2022). Big Data for Decision Making. Lecture examples and exercises. (Version 1.0.0). URL: https://github.com/vpekar/bd4dm

In [ ]:

```
```