Individual Assignment Submission - 220273880

ROAD CASUALITY ANALYSIS

Table of content

Introduction

For this portfolio, I will be analysing the statistics on traffic safety in the UK.This will be done by utilising the databases for statistics on car casualties and statistics on casualties from accidents.

Objective

Our objective is to predict the number of fatalities by combining two datasets.we have considered Two databases for the study from our group assignment. We need to also consider a few other important factors that are essential for performing a comprehensive analysis. This will benefit the Emergency Management Agency because fewer mishaps will happen as a result of such events, and they may even be able to lower the number of fatalities. We are addressing our regression issue by choosing the essential elements as our models and even investigating the feature significance of these variables.

Target Variable = Number of casualities

In [23]:

```
# Importing Libraries for our analysis

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import cross_val_score, RandomizedSearchCV
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
```

In [25]:

```
# Loading the excel file
```

```
df_train = pd.read_excel('C:\\Users\\Vijay\\Downloads\\trainset1.xlsx')
df_test = pd.read_excel('C:\\Users\\Vijay\\Downloads\\testset1.xlsx')
```

In [26]:

```
print(df_train.shape)
print(df_test.shape)
```

```
(5170, 16)
(1302, 16)
```

There are 5170 Rows and 16 Columns in our Training set and 1302 Rows and 16 Column in out Testing set

In [5]:

```
df_train.head()
```

Out[5]:

	Unnamed: 0	number_of_vehicles	sex_of_driver	urban_or_rural_area	light_conditions	road_type	vehicle_left_hand_drive	first_
0	190	2	1	1	4	6	1	
1	5367	2	1	1	4	3	2	
2	6708	7	2	1	1	7	1	
3	5238	2	2	1	7	6	9	
4	3349	1	1	1	4	6	9	

In [28]:

```
# checking the value counts of accident severity in the training dataset

df_train['accident_sev'].value_counts()
```

Out[28]:

```
Slight      4525
Serious      613
Fatal         32
Name: accident_sev, dtype: int64
```

In [31]:

```
# converting the categorical feature levels into numerical values

le = LabelEncoder()

df_train['accident_sev'] = le.fit_transform(df_train['accident_sev'])
df_test['accident_sev'] = le.fit_transform(df_test['accident_sev'])
```

Data Preprocessing

Every data analysis project must start with data preprocessing. Data must be cleaned, transformed, and made ready for analysis.

In [8]:

```
x_train = df_train.drop(['number_of_casualties'], axis=1)
y_train = df_train['number_of_casualties']

x_test = df_test.drop(['number_of_casualties'], axis=1)
y_test = df_test['number_of_casualties']
```

Now, We will be scaling using Standard Scaler to remove the mean and scales each feature/variable to unit variance

In [32]:

```
# Scaling using Standard Scaler
scaler = StandardScaler()

x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

Baseline Model

A baseline model is a straightforward model that is used as a basis or a point of comparison when evaluating the performance of more intricate models.

In [34]:

```
#Calculating mean value training set
mean_rating=y_train.mean()
mean_rating
```

Out[34]:

```
1.2415860735009672
```

In [37]:

```
#Calculating RMSE for baseline model
from sklearn.metrics import mean_squared_error

yhat = np.full((y_train.shape[0], 1), mean_rating)
baseline_mse = mean_squared_error(y_train, yhat)


baseline_rmse = np.sqrt(baseline_mse)
baseline_rmse
```

Out[37]:

```
0.6783839874787445
```

We got the RMSE value of our baseline model as 0.6783839874787445

Models for analysis

After our baseline model, We will be running out dataset on various models to get the initial Analysis results.

Linear Regression Model

In [10]:

```
# create linear regression object
reg = LinearRegression()

reg.fit(x_train_scaled, y_train)

# print coefficients in a dataframe
coef_df = pd.DataFrame({'Attribute' : x_train.columns, 'Coefficient': reg.coef_})
display(coef_df)
print('Intercept: ',reg.intercept_)

# calculate R squared value
y_pred = reg.predict(x_test_scaled)
r2 = r2_score(y_test, y_pred)
print('R squared value:', r2)
```

```python
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# compute root mean squared error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

	Attribute	Coefficient
0	Unnamed: 0	-0.026371
1	number_of_vehicles	0.142787
2	sex_of_driver	0.008326
3	urban_or_rural_area	0.028676
4	light_conditions	0.009722
5	road_type	0.000549
6	vehicle_left_hand_drive	0.008356
7	first_point_of_impact	-0.015179
8	age_band_of_driver	-0.076492
9	age_of_vehicle	0.022937
10	driver_home_area_type	-0.007916
11	vehicle_manoeuvre	-0.037507
12	road_surface_conditions	-0.002436
13	age_of_driver	0.078078
14	accident_sev	-0.054082

```
Intercept:  1.2415860735009672
R squared value: 0.05235384928332165
Mean Squared Error (MSE): 0.4499899874293023
Root Mean Squared Error (RMSE): 0.6708129302788537
```

Decision Tree Regressor

In [11]:

```python
regressor = DecisionTreeRegressor(random_state=0)

regressor.fit(x_train, y_train)

# predict the output
y_pred = regressor.predict(x_test)

r2 = r2_score(y_test, y_pred)
print('R squared value:', r2)

# compute mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# compute root mean squared error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)
```

```
R squared value: -0.6368650679281664
Mean Squared Error (MSE): 0.7772657450076805
Root Mean Squared Error (RMSE): 0.8816267606009249
```

KNN Regressor

In [12]:

```
regressor = KNeighborsRegressor(n_neighbors=3)

regressor.fit(x_train, y_train)

y_pred = regressor.predict(x_test)

# compute mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# compute root mean squared error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

# compute R2 score
r2 = r2_score(y_test, y_pred)
print("R2 Score:", r2)
```

```
Mean Squared Error (MSE): 0.5455709165386585
Root Mean Squared Error (RMSE): 0.7386277252707608
R2 Score: -0.14893262837777432
```

Random Forest Regressor

In [13]:

```
regressor = RandomForestRegressor(n_estimators=100, random_state=0)

regressor.fit(x_train, y_train)

y_pred = regressor.predict(x_test)

# compute mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# compute root mean squared error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

# compute R2 score
r2 = r2_score(y_test, y_pred)
print("R2 Score:", r2)
```

```
Mean Squared Error (MSE): 0.3830470814132105
Root Mean Squared Error (RMSE): 0.6189079749148579
R2 Score: 0.19333073538326973
```

Cross Validation

Cross-validation is a technique for evaluating the performance of a machine learning model on a dataset

So, we will be checking our dataset using cross validation for further steps

Linear Regressor

In [61]:

```
regressor = LinearRegression()

# perform 5-fold cross-validation on training data
scores = cross_val_score(regressor, x_train, y_train, cv=5)

print("Cross-Validation Scores:", scores)
```

```
regressor.fit(x_train, y_train)

y_pred = regressor.predict(x_test)

# compute mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# compute root mean squared error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

# compute R2 score
r2 = r2_score(y_test, y_pred)
print("R2 Score:", r2)
```

```
Cross-Validation Scores: [ 0.07930346  0.08364844  0.05099385  0.02098083 -0.01303251]
Mean Squared Error (MSE): 0.4485223256015775
Root Mean Squared Error (RMSE): 0.6697180941273556
R2 Score: 0.055444638235188526
```

Decision Tree Regressor

In [15]:

```
regressor = DecisionTreeRegressor(random_state=42)

scores = cross_val_score(regressor, x_train, y_train, cv=5)

print("Cross-Validation Scores:", scores)

regressor.fit(x_train, y_train)

y_pred = regressor.predict(x_test)

mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

r2 = r2_score(y_test, y_pred)
print("R2 Score:", r2)
```

```
Cross-Validation Scores: [-0.41908814 -0.32219143 -0.79471046 -0.86611738 -0.77030983]
Mean Squared Error (MSE): 0.7795698924731183
Root Mean Squared Error (RMSE): 0.8829325526183289
R2 Score: -0.6417174347303249
```

KNN Regressor

In [16]:

```
regressor = KNeighborsRegressor()

scores = cross_val_score(regressor, x_train, y_train, cv=5)

# print the cross-validation scores
print("Cross-Validation Scores:", scores)

# fit the model with training data
regressor.fit(x_train, y_train)

# predict the output for testing data
y_pred = regressor.predict(x_test)

# compute mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
```

```
print("Mean Squared Error (MSE):", mse)

# compute root mean squared error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

# compute R2 score
r2 = r2_score(y_test, y_pred)
print("R2 Score:", r2)
```

```
Cross-Validation Scores: [-0.05111807  0.05213794 -0.04171725 -0.02559309 -0.16252128]
Mean Squared Error (MSE): 0.5232565284178187
Root Mean Squared Error (RMSE): 0.7233647271037057
R2 Score: -0.1019401516582028
```

Hyper Parameter Tuning And Model Evaluation

Hyperparameter tuning is the process of selecting the optimal hyperparameters for a machine learning model.

Now, We will be Working on the following models by using different hyper parameters until we obtain the best model

For a Good regression model, The criteria on which we check our analysis and results is RMSE - Root Mean Squared Error value. The best model is considerd to have a least RMSE values.

Linear Regression

In [17]:

```
regressor = LinearRegression()

# set up the parameter grid for RandomizedSearchCV
param_grid = {'fit_intercept': [True, False],
              'positive': [True, False],
              'copy_X': [True, False]}

# perform Randomized Search CV with 5-fold cross-validation on training data
rand_search = RandomizedSearchCV(regressor, param_grid, cv=5, n_iter=10, random_state=0)
rand_search.fit(x_train, y_train)

# print the best parameters and best score
print("Best Parameters:", rand_search.best_params_)
print("Best Score:", rand_search.best_score_)

# fit the model with training data using the best parameters
regressor = LinearRegression(**rand_search.best_params_)
regressor.fit(x_train, y_train)

# predict the output for testing data
y_pred = regressor.predict(x_test)

# compute mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# compute root mean squared error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

# compute R2 score
r2 = r2_score(y_test, y_pred)
print("R2 Score:", r2)
```

```
Best Parameters: {'positive': False, 'fit_intercept': True, 'copy_X': True}
Best Score: 0.04579804516851567
Mean Squared Error (MSE): 0.4499899874293023
```

```
Root Mean Squared Error (RMSE): 0.6708129302788537
R2 Score: 0.05235384928332165
```

For our Regression model, The value of R2 for the following parameters is 0.052, mean 5% which is very less indicating that the model is not that good and the RMSE value is 0.6708

Decision Tree Regressor

In [18]:

```
regressor = DecisionTreeRegressor(random_state=0)

param_grid = {'max_depth': [1, 2, 3, 4, 5],
              'min_samples_split': [2, 3, 4],
              'min_samples_leaf': [1, 2, 3],
              'max_features': ['auto', 'sqrt', 'log2']}

rand_search = RandomizedSearchCV(regressor, param_grid, cv=5, n_iter=10, random_state=0)
rand_search.fit(x_train, y_train)

# print the best parameters and best score
print("Best Parameters:", rand_search.best_params_)
print("Best Score:", rand_search.best_score_)

# fit the model with training data using the best parameters
regressor = DecisionTreeRegressor(**rand_search.best_params_, random_state=0)
regressor.fit(x_train, y_train)

# predict the output for testing data
y_pred = regressor.predict(x_test)

# compute mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# compute root mean squared error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

# compute R2 score
r2 = r2_score(y_test, y_pred)
print("R2 Score:", r2)
```

```
Best Parameters: {'min_samples_split': 4, 'min_samples_leaf': 2, 'max_features': 'auto',
'max_depth': 5}
Best Score: 0.1608338350465658
Mean Squared Error (MSE): 0.3871416909807951
Root Mean Squared Error (RMSE): 0.6222071126086515
R2 Score: 0.1847077857537096
```

For our Decision Tress, The RMSE value is 0.62.

KNN Regressor

In [19]:

```
regressor = KNeighborsRegressor()

# set up the parameter grid for RandomizedSearchCV
param_grid = {'n_neighbors': [3, 5, 7, 9],
              'weights': ['uniform', 'distance'],
              'algorithm': ['ball_tree', 'kd_tree', 'brute']}

# perform Randomized Search CV with 5-fold cross-validation on training data
rand_search = RandomizedSearchCV(regressor, param_grid, cv=5, n_iter=10, random_state=0)
rand_search.fit(x_train, y_train)
```

```
# print the best parameters and best score
print("Best Parameters:", rand_search.best_params_)
print("Best Score:", rand_search.best_score_)

# fit the model with training data using the best parameters
regressor = KNeighborsRegressor(**rand_search.best_params_)
regressor.fit(x_train, y_train)

# predict the output for testing data
y_pred = regressor.predict(x_test)

# compute mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# compute root mean squared error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

# compute R2 score
r2 = r2_score(y_test, y_pred)
print("R2 Score:", r2)
```

```
Best Parameters: {'weights': 'distance', 'n_neighbors': 9, 'algorithm': 'brute'}
Best Score: 0.0157692325594768
Mean Squared Error (MSE): 0.4817461428954608
Root Mean Squared Error (RMSE): 0.6940793491348527
R2 Score: -0.014522302030586243
```

The RMSE value for KNN is 0.69

Random Forest Regressor

In [20]:

```
regressor = RandomForestRegressor()

# set up the parameter grid for RandomizedSearchCV
param_grid = {'n_estimators': [100, 200, 300, 400, 500],
              'max_features': ['auto', 'sqrt'],
              'max_depth': [5, 10, 15, 20, 25, None],
              'min_samples_split': [2, 5, 10],
              'min_samples_leaf': [1, 2, 4],
              'bootstrap': [True, False]}

# perform Randomized Search CV with 5-fold cross-validation on training data
rand_search = RandomizedSearchCV(regressor, param_grid, cv=5, n_iter=10, random_state=0)
rand_search.fit(x_train, y_train)

# print the best parameters and best score
print("Best Parameters:", rand_search.best_params_)
print("Best Score:", rand_search.best_score_)

# fit the model with training data using the best parameters
regressor = RandomForestRegressor(**rand_search.best_params_)
regressor.fit(x_train, y_train)

# predict the output for testing data
y_pred = regressor.predict(x_test)

# compute mean squared error (MSE)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error (MSE):", mse)

# compute root mean squared error (RMSE)
rmse = np.sqrt(mse)
print("Root Mean Squared Error (RMSE):", rmse)

# compute R2 score
```

```
r2 = r2_score(y_test, y_pred)
print("R2 Score:", r2)
```

```
Best Parameters: {'n_estimators': 500, 'min_samples_split': 5, 'min_samples_leaf': 1, 'ma
x_features': 'sqrt', 'max_depth': 10, 'bootstrap': False}
Best Score: 0.15657900244374148
Mean Squared Error (MSE): 0.3830603679949002
Root Mean Squared Error (RMSE): 0.6189187087129457
R2 Score: 0.1933027548095988
```

The Value of Root mean square error for Random Forest Regressor is 0.61

Conclusion

From the Analysis that we have made above. The least RMSE value is for the Random forest regression model with RMSE value of 0.6189187087129457, Which is lowest amongst all the models that we have run after hyperparameter tuning.

Also, there is no overfitting in the model.

Although the RMSE value for our best model is 0.61 which is very low, This model can not be possible in the real world as it is practically not possible to have this low value. However, This can made possible by taking more data inputs so to clear image and understanding.