Appendix_C_LogReg_and_k-NN_modeling

August 10, 2024

1 Logistic Regression and k-Nearest Neighbors Modeling

This notebook contains the code and details for logistic regression and k-nearest neighbors modeling.

[ ]:
```
import warnings
warnings.filterwarnings('ignore')
```

[ ]:
```
import pandas as pd
import numpy as np

from imblearn.over_sampling import RandomOverSampler, SMOTE
from imblearn.under_sampling import RandomUnderSampler
from pathlib import Path
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
```

[ ]:
```
# Import training dataset
dataset = Path('../dataset')
df = pd.read_csv(dataset/'accidents_clean_train.csv')
df.head()
```

[ ]:
```
  Area_accident_occured Types_of_Junction       Light_conditions  \
0     Residential areas       No junction              Daylight
1          Office areas       No junction              Daylight
2    Recreational areas       No junction              Daylight
3          Office areas           Y Shape  Darkness - lights lit
4      Industrial areas           Y Shape  Darkness - lights lit

   Number_of_vehicles_involved  Number_of_casualties  \
0                            2                     2
1                            2                     2
2                            2                     2
```

```
3                            2                    2
4                            2                    2

            Cause_of_accident Day_of_week Sex_of_driver Age_band_of_driver  \
0             Moving Backward      Monday          Male              18-30
1                   Overtaking      Monday          Male              31-50
2   Changing lane to the left      Monday          Male              18-30
3  Changing lane to the right      Sunday          Male              18-30
4                   Overtaking      Sunday          Male              18-30

  Accident_severity
0     Slight Injury
1     Slight Injury
2    Serious Injury
3     Slight Injury
4     Slight Injury
```

[ ]:
```
# Get feature columns
columns = df.columns.tolist()
print(columns)

features = ['Area_accident_occured', 'Types_of_Junction', 'Light_conditions',␣
 ↪'Number_of_vehicles_involved', 'Number_of_casualties', 'Cause_of_accident',␣
 ↪'Day_of_week', 'Sex_of_driver', 'Age_band_of_driver']

target = 'Accident_severity'
```

```
['Area_accident_occured', 'Types_of_Junction', 'Light_conditions',
'Number_of_vehicles_involved', 'Number_of_casualties', 'Cause_of_accident',
'Day_of_week', 'Sex_of_driver', 'Age_band_of_driver', 'Accident_severity']
```

1.1 Convert features to categorical

[ ]:
```
# Convert to categorical
X = df[features]
X = pd.get_dummies(X, drop_first=True)
X.head()
```

[ ]:
```
   Number_of_vehicles_involved  Number_of_casualties  \
0                            2                     2
1                            2                     2
2                            2                     2
3                            2                     2
4                            2                     2

   Area_accident_occured_  Recreational areas  \
0                                       False
```

```
1                                    False
2                                     True
3                                    False
4                                    False

   Area_accident_occured_ Church areas  Area_accident_occured_ Hospital areas  \
0                               False                                  False
1                               False                                  False
2                               False                                  False
3                               False                                  False
4                               False                                  False

   Area_accident_occured_ Industrial areas  \
0                                   False
1                                   False
2                                   False
3                                   False
4                                    True

   Area_accident_occured_ Outside rural areas  \
0                                     False
1                                     False
2                                     False
3                                     False
4                                     False

   Area_accident_occured_Office areas  Area_accident_occured_Other  \
0                               False                        False
1                                True                        False
2                               False                        False
3                                True                        False
4                               False                        False

   Area_accident_occured_Recreational areas  …  Day_of_week_Sunday  \
0                                     False  …               False
1                                     False  …               False
2                                     False  …               False
3                                     False  …                True
4                                     False  …                True

   Day_of_week_Thursday  Day_of_week_Tuesday  Day_of_week_Wednesday  \
0                 False                False                  False
1                 False                False                  False
2                 False                False                  False
3                 False                False                  False
4                 False                False                  False
```

```
   Sex_of_driver_Male  Sex_of_driver_Unknown  Age_band_of_driver_31-50  \
0                True                  False                     False
1                True                  False                      True
2                True                  False                     False
3                True                  False                     False
4                True                  False                     False

   Age_band_of_driver_Over 51  Age_band_of_driver_Under 18  \
0                       False                        False
1                       False                        False
2                       False                        False
3                       False                        False
4                       False                        False

   Age_band_of_driver_Unknown
0                       False
1                       False
2                       False
3                       False
4                       False

[5 rows x 56 columns]
```

1.2 Create mapped columns of target variable

[ ]:
```
# Create mapped columns of target
df['Accident_slight'] = df[target].map(
    {'Slight Injury': 0}
    ).fillna(1).astype(int)
df['Accident_serious'] = df[target].map(
    {'Serious Injury': 0}
    ).fillna(1).astype(int)
df['Accident_severity_mapped'] = df[target].map({
    'Serious Injury': 0,
    'Slight Injury': 1,
    'Fatal injury': 2
    })

df.head()
```

[ ]:
```
  Area_accident_occured Types_of_Junction       Light_conditions  \
0     Residential areas       No junction              Daylight
1          Office areas       No junction              Daylight
2    Recreational areas       No junction              Daylight
3          Office areas           Y Shape  Darkness - lights lit
4      Industrial areas           Y Shape  Darkness - lights lit
```

```
   Number_of_vehicles_involved  Number_of_casualties  \
0                            2                     2
1                            2                     2
2                            2                     2
3                            2                     2
4                            2                     2

            Cause_of_accident Day_of_week Sex_of_driver Age_band_of_driver  \
0             Moving Backward      Monday          Male              18-30
1                   Overtaking      Monday          Male              31-50
2   Changing lane to the left      Monday          Male              18-30
3  Changing lane to the right      Sunday          Male              18-30
4                   Overtaking      Sunday          Male              18-30

  Accident_severity  Accident_slight  Accident_serious  \
0     Slight Injury                0                 1
1     Slight Injury                0                 1
2    Serious Injury                1                 0
3     Slight Injury                0                 1
4     Slight Injury                0                 1

   Accident_severity_mapped
0                         1
1                         1
2                         0
3                         1
4                         1
```

[ ]:
```
df.shape
```

[ ]:
```
(8210, 13)
```

1.3 Create y_test variables for ML training

[ ]:
```
y = df[target]
y_mapped = df['Accident_severity_mapped']
y_slight = df['Accident_slight']
y_serious = df['Accident_serious']
```

1.4 Resampling

[ ]:
```
# Standardize the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

[ ]:
```
oversample = RandomOverSampler(random_state=42)
```

```
X_resampled, y_resampled = oversample.fit_resample(X, y_mapped)
```

[ ]:
```
# Function to print evaluation metrics
def print_evaluation_metrics(test, pred):
    print("Accuracy: ", accuracy_score(test, pred))
    print("Confusion Matrix:\n", confusion_matrix(test, pred))
    print("Classification Report:\n", classification_report(test, pred))

def print_cross_val_scores(scores):
    print("Cross-validation scores: ", scores)
    print("Average score: ", scores.mean())
```

[ ]:
```

```

2 Linear Regression

2.1 Slight Injury (1) vs. Serious Injury (0) vs. Fatal Injury (2)

[ ]:
```
#Split testing data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_mapped,␣
 ↪test_size=0.3, random_state=42)

#Train model
log_reg_mapped = LogisticRegression(max_iter=1000, multi_class='multinomial',␣
 ↪solver='lbfgs')
log_reg_mapped.fit(X_train, y_train)

#Create prediction
y_pred_mapped = log_reg_mapped.predict(X_test)

print("Logistic Regression (Slight Injury vs. Serious Injury vs. Fatal Injury):
 ↪")
print_evaluation_metrics(y_test, y_pred_mapped)
score_mapped = cross_val_score(log_reg_mapped, X_test, y_test, cv=5)
print_cross_val_scores(score_mapped)
```

```
Logistic Regression (Slight Injury vs. Serious Injury vs. Fatal Injury):
Accuracy:  0.8574908647990256
Confusion Matrix:
 [[   1  329    0]
 [   0 2111    0]
 [   0   22    0]]
Classification Report:
              precision    recall  f1-score   support

           0       1.00      0.00      0.01       330
           1       0.86      1.00      0.92      2111
```

```
           2       0.00      0.00      0.00        22

    accuracy                           0.86      2463
   macro avg       0.62      0.33      0.31      2463
weighted avg       0.87      0.86      0.79      2463

Cross-validation scores:  [0.85598377 0.85192698 0.85395538 0.85772358
0.85772358]
Average score:  0.8554626560464387
```

##Slight Injury (0) vs. Serious Injury/Fatal Injury (1)

[ ]:
```
# Split testing data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_slight,␣
 ↪test_size=0.3, random_state=42)


# Train model
log_reg_slight = LogisticRegression(max_iter=1000)
log_reg_slight.fit(X_train, y_train)

# Create prediction
y_pred_slight = log_reg_slight.predict(X_test)

print("Logistic Regression (Slight Injury vs. Serious/Fatal Injury):")
print_evaluation_metrics(y_test, y_pred_slight)
score_slight = cross_val_score(log_reg_slight, X_test, y_test, cv=5)
print_cross_val_scores(score_slight)
```

```
Logistic Regression (Slight Injury vs. Serious/Fatal Injury):
Accuracy:  0.8574908647990256
Confusion Matrix:
 [[2111    0]
 [ 351    1]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      2111
           1       1.00      0.00      0.01       352

    accuracy                           0.86      2463
   macro avg       0.93      0.50      0.46      2463
weighted avg       0.88      0.86      0.79      2463

Cross-validation scores:  [0.85598377 0.85598377 0.85395538 0.8597561
0.85772358]
Average score:  0.8566805191378485
```

##Slight/Fatal Injury (1) vs. Serious Injury (0)

[ ]:
```
#Split testing data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_serious,␣
 ↪test_size=0.3, random_state=42)

#Train model
log_reg_serious = LogisticRegression(max_iter=1000)
log_reg_serious.fit(X_train, y_train)

#Create prediction
y_pred_serious = log_reg_serious.predict(X_test)
print("Logistic Regression (Slight/Fatal Injury vs. Serious Injury):")
print_evaluation_metrics(y_test, y_pred_serious)

score_serious = cross_val_score(log_reg_serious, X_test, y_test, cv=5)
print_cross_val_scores(score_serious)
```

```
Logistic Regression (Slight/Fatal Injury vs. Serious Injury):
Accuracy:  0.8660170523751523
Confusion Matrix:
 [[   0  330]
 [   0 2133]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       330
           1       0.87      1.00      0.93      2133

    accuracy                           0.87      2463
   macro avg       0.43      0.50      0.46      2463
weighted avg       0.75      0.87      0.80      2463

Cross-validation scores:  [0.86409736 0.86409736 0.86206897 0.86788618
0.86585366]
Average score:  0.8648007058163889
```

3 Resample Target Data

3.1 Slight Injury (1) vs. Serious Injury (0) vs. Fatal Injury (2)

[ ]:
```
# Apply RandomOverSampler
oversample = RandomOverSampler(random_state=42)

X_resampled, y_resampled = oversample.fit_resample(X_scaled, y_mapped)

# Check distribution
print(y_resampled.value_counts())
```

```
Accident_severity_mapped
```

```
1    7082
0    7082
2    7082
Name: count, dtype: int64
```

[ ]:
```
#Split testing data
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
 ↪test_size=0.3, random_state=42)

#Train model
log_reg_mapped = LogisticRegression(max_iter=1000, multi_class='multinomial',␣
 ↪solver='lbfgs')
log_reg_mapped.fit(X_train, y_train)

#Create prediction
y_pred_mapped = log_reg_mapped.predict(X_test)


print("Logistic Regression with Oversampling (Slight Injury vs. Serious Injury␣
 ↪vs. Fatal Injury):")
print_evaluation_metrics(y_test, y_pred_mapped)
score_mapped = cross_val_score(log_reg_mapped, X_test, y_test, cv=5)
print_cross_val_scores(score_mapped)
```

```
Logistic Regression with Oversampling (Slight Injury vs. Serious Injury vs.
Fatal Injury):
Accuracy:  0.5527141512394101
Confusion Matrix:
 [[ 851  797  519]
 [ 630 1050  436]
 [ 291  178 1622]]
Classification Report:
              precision    recall  f1-score   support

           0       0.48      0.39      0.43      2167
           1       0.52      0.50      0.51      2116
           2       0.63      0.78      0.69      2091

    accuracy                           0.55      6374
   macro avg       0.54      0.55      0.54      6374
weighted avg       0.54      0.55      0.54      6374

Cross-validation scores:  [0.54666667 0.52941176 0.54823529 0.53882353
0.51020408]
Average score:  0.5346682673069227
```

##Slight Injury (0) vs. Serious Injury/Fatal Injury (1)

[ ]:
```
X_resampled, y_resampled = oversample.fit_resample(X_scaled, y_slight)

#Check distribution
print(y_resampled.value_counts())
```

```
Accident_slight
0    7082
1    7082
Name: count, dtype: int64
```

[ ]:
```
#Split testing data
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
 ↪test_size=0.3, random_state=42)

#Train model
log_reg_slight = LogisticRegression(max_iter=1000)
log_reg_slight.fit(X_train, y_train)

#Create predictions
y_pred_slight = log_reg_slight.predict(X_test)


print("Logistic Regression with Oversampling (Slight Injury vs. Serious/Fatal␣
 ↪Injury):")
print_evaluation_metrics(y_test, y_pred_slight)
score_slight = cross_val_score(log_reg_slight, X_test, y_test, cv=5)
print_cross_val_scores(score_slight)
```

```
Logistic Regression with Oversampling (Slight Injury vs. Serious/Fatal Injury):
Accuracy:  0.5854117647058823
Confusion Matrix:
 [[1217  914]
 [ 848 1271]]
Classification Report:
              precision    recall  f1-score   support

           0       0.59      0.57      0.58      2131
           1       0.58      0.60      0.59      2119

    accuracy                           0.59      4250
   macro avg       0.59      0.59      0.59      4250
weighted avg       0.59      0.59      0.59      4250

Cross-validation scores:  [0.57411765 0.55647059 0.55882353 0.61058824
0.55764706]
Average score:  0.5715294117647058
```

##Slight/Fatal Injury (1) vs. Serious Injury (0)

[ ]:
```
X_resampled, y_resampled = oversample.fit_resample(X_scaled, y_serious)

#Check distribution
print(y_resampled.value_counts())
```

```
Accident_serious
1    7164
0    7164
Name: count, dtype: int64
```

[ ]:
```
#Split testing data
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
 ↪test_size=0.3, random_state=42)

#Train model
log_reg_serious = LogisticRegression(max_iter=1000)
log_reg_serious.fit(X_train, y_train)

#Create prediction
y_pred_serious = log_reg_serious.predict(X_test)


print("Logistic Regression with Oversampling (Slight/Fatal Injury vs. Serious␣
 ↪Injury):")
print_evaluation_metrics(y_test, y_pred_serious)
score_serious = cross_val_score(log_reg_serious, X_test, y_test, cv=5)
print_cross_val_scores(score_serious)
```

```
Logistic Regression with Oversampling (Slight/Fatal Injury vs. Serious Injury):
Accuracy:  0.5782740172133054
Confusion Matrix:
 [[1269  896]
 [ 917 1217]]
Classification Report:
              precision    recall  f1-score   support

           0       0.58      0.59      0.58      2165
           1       0.58      0.57      0.57      2134

    accuracy                           0.58      4299
   macro avg       0.58      0.58      0.58      4299
weighted avg       0.58      0.58      0.58      4299

Cross-validation scores:  [0.57674419 0.58255814 0.53953488 0.58023256
0.57159488]
Average score:  0.5701329290413407
```

3.1.1 Logistic Regression Summary

Target: Slight Injury (1) vs. Serious Injury (0) vs. Fatal Injury (2)

Without Oversampling:

Accuracy: 85.74%

Precision Serious Injury: 86%

Recall: 100%

With Oversampling:

Accuracy: 57.87%

Precision Serious Injury: 58%

Recall: 57%

Target: Slight Injury (0) vs. Serious Injury/Fatal Injury (1)

Without Oversampling:

Accuracy: 85.75%

Precision Serious Injury: 100%

Recall: 0%

With Oversampling:

Accuracy: 57.87%

Precision Serious Injury: 58%

Recall: 57%

Target: Slight/Fatal Injury (1) vs. Serious Injury (0)

Without Oversampling:

Accuracy: 86.60%

Precision Serious Injury: 87%

Recall: 100%

With Oversampling:

Accuracy: 57.87%

Precision Serious Injury: 58%

Recall: 57%

4 KNN Model

[ ]:
```
param_grid = {'n_neighbors': range(1, 50)}
```

4.1 Without Oversampling

4.2 Slight Injury (1) vs. Serious Injury (0) vs. Fatal Injury (2)

[ ]:
```
#Split testing data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_mapped,␣
 ↪test_size=0.3, random_state=42)

#Train model with best number of neighbors
knn_mapped = KNeighborsClassifier()
grid_search_mapped = GridSearchCV(knn_mapped, param_grid, cv=5,␣
 ↪scoring='accuracy')
grid_search_mapped.fit(X_train, y_train)
best_knn_mapped = grid_search_mapped.best_estimator_

#Create prediction
y_pred_mapped = best_knn_mapped.predict(X_test)

print("Best number of neighbors:", grid_search_mapped.
 ↪best_params_['n_neighbors'])
print("KNN (Slight Injury vs. Serious Injury vs. Fatal Injury):")
print_evaluation_metrics(y_test, y_pred_mapped)
score_mapped = cross_val_score(knn_mapped, X_test, y_test, cv=5)
print_cross_val_scores(score_mapped)
```

```
Best number of neighbors: 21
KNN (Slight Injury vs. Serious Injury vs. Fatal Injury):
Accuracy:  0.857084855866829
Confusion Matrix:
 [[   0  330    0]
 [   0 2111    0]
 [   0   22    0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       330
           1       0.86      1.00      0.92      2111
           2       0.00      0.00      0.00        22

    accuracy                           0.86      2463
   macro avg       0.29      0.33      0.31      2463
weighted avg       0.73      0.86      0.79      2463

Cross-validation scores:  [0.84381339 0.86004057 0.81947262 0.83739837 0.8495935
```

```
]
Average score:  0.8420636883853625
```

##Slight Injury (0) vs. Serious Injury/Fatal Injury (1)

[ ]:
```
#Split testing data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_slight,␣
 ↪test_size=0.3, random_state=42)

#Train model with best number of neighbors
knn_slight = KNeighborsClassifier()
grid_search_slight = GridSearchCV(knn_slight, param_grid, cv=5,␣
 ↪scoring='accuracy')
grid_search_slight.fit(X_train, y_train)
best_knn_slight = grid_search_slight.best_estimator_

#Create prediction
y_pred_slight = best_knn_slight.predict(X_test)

print("Best number of neighbors for Slight Injury vs. Serious/Fatal Injury:",␣
 ↪grid_search_slight.best_params_['n_neighbors'])
print("KNN (Slight Injury vs. Serious/Fatal Injury):")
print_evaluation_metrics(y_test, y_pred_slight)
score_slight = cross_val_score(best_knn_slight, X_test, y_test, cv=5)
print_cross_val_scores(score_slight)
```

```
Best number of neighbors for Slight Injury vs. Serious/Fatal Injury: 20
KNN (Slight Injury vs. Serious/Fatal Injury):
Accuracy:  0.857084855866829
Confusion Matrix:
 [[2111    0]
 [ 352    0]]
Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.92      2111
           1       0.00      0.00      0.00       352

    accuracy                           0.86      2463
   macro avg       0.43      0.50      0.46      2463
weighted avg       0.73      0.86      0.79      2463

Cross-validation scores:  [0.85801217 0.85598377 0.85598377 0.85772358
0.85772358]
Average score:  0.8570853740991771
```

##Slight/Fatal Injury (1) vs. Serious Injury (0)

[ ]:
```
#Split testing data
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_serious,␣
 ↪test_size=0.3, random_state=42)

#Train model with best number of neighbors
knn_serious = KNeighborsClassifier()
grid_search_serious = GridSearchCV(knn_serious, param_grid, cv=5,␣
 ↪scoring='accuracy')
grid_search_serious.fit(X_train, y_train)
best_knn_serious = grid_search_serious.best_estimator_

#Create prediction
y_pred_serious = best_knn_serious.predict(X_test)

print("Best number of neighbors for Slight/Fatal Injury vs. Serious Injury:",␣
 ↪grid_search_serious.best_params_['n_neighbors'])
print("KNN (Slight/Fatal Injury vs. Serious Injury):")
print_evaluation_metrics(y_test, y_pred_serious)
score_serious = cross_val_score(best_knn_serious, X_test, y_test, cv=5)
print_cross_val_scores(score_serious)
```

```
Best number of neighbors for Slight/Fatal Injury vs. Serious Injury: 21
KNN (Slight/Fatal Injury vs. Serious Injury):
Accuracy:  0.8660170523751523
Confusion Matrix:
 [[   0  330]
 [   0 2133]]
Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00       330
           1       0.87      1.00      0.93      2133

    accuracy                           0.87      2463
   macro avg       0.43      0.50      0.46      2463
weighted avg       0.75      0.87      0.80      2463

Cross-validation scores:  [0.86612576 0.86612576 0.86409736 0.86585366
0.86585366]
Average score:  0.8656112402909019
```

[ ]:
```

```

5 Resample Target Data

5.1 Slight Injury (1) vs. Serious Injury (0) vs. Fatal Injury (2)

[ ]:
```
# Apply RandomOverSampler
oversample = RandomOverSampler(random_state=42)

X_resampled, y_resampled = oversample.fit_resample(X_scaled, y_mapped)

# Check distribution
print(y_resampled.value_counts())
```

```
Accident_severity_mapped
1    7082
0    7082
2    7082
Name: count, dtype: int64
```

[ ]:
```
#Split testing data
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
 ↪test_size=0.3, random_state=42)

#Train model with best number of neighbors
knn_mapped = KNeighborsClassifier()
grid_search_mapped = GridSearchCV(knn_mapped, param_grid, cv=5,␣
 ↪scoring='accuracy')
grid_search_mapped.fit(X_train, y_train)
best_knn_mapped = grid_search_mapped.best_estimator_

#Create prediction
y_pred_mapped = best_knn_mapped.predict(X_test)


print("Best number of neighbors for Slight Injury vs. Serious Injury vs. Fatal␣
 ↪Injury (oversampled):", grid_search_mapped.best_params_['n_neighbors'])
print("KNN with Oversampling (Slight Injury vs. Serious Injury vs. Fatal␣
 ↪Injury):")
print_evaluation_metrics(y_test, y_pred_mapped)
score_mapped = cross_val_score(best_knn_mapped, X_test, y_test, cv=5)
print_cross_val_scores(score_mapped)
```

```
Best number of neighbors for Slight Injury vs. Serious Injury vs. Fatal Injury
(oversampled): 1
KNN with Oversampling (Slight Injury vs. Serious Injury vs. Fatal Injury):
Accuracy:  0.9237527455287103
Confusion Matrix:
 [[2090   70    7]
 [ 356 1737   23]
```

```
 [   0   30 2061]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.96      0.91      2167
           1       0.95      0.82      0.88      2116
           2       0.99      0.99      0.99      2091

    accuracy                           0.92      6374
   macro avg       0.93      0.92      0.92      6374
weighted avg       0.93      0.92      0.92      6374

Cross-validation scores:  [0.84       0.84784314 0.84078431 0.83215686
0.83516484]
Average score:  0.8391898297780651
```

##Slight Injury (0) vs. Serious Injury/Fatal Injury (1)

[ ]:
```
X_resampled, y_resampled = oversample.fit_resample(X_scaled, y_slight)

#Check distribution
print(y_resampled.value_counts())
```

```
Accident_slight
0    7082
1    7082
Name: count, dtype: int64
```

[ ]:
```
#Split testing data
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
 ↪test_size=0.3, random_state=42)

#Train model with best number of neighbors
knn_slight = KNeighborsClassifier()
grid_search_slight = GridSearchCV(knn_slight, param_grid, cv=5,␣
 ↪scoring='accuracy')
grid_search_slight.fit(X_train, y_train)
best_knn_slight = grid_search_slight.best_estimator_

#Create prediction
y_pred_slight = best_knn_slight.predict(X_test)

print("Best number of neighbors for Slight Injury vs. Serious/Fatal Injury␣
 ↪(oversampled):", grid_search_slight.best_params_['n_neighbors'])
print("KNN with Oversampling (Slight Injury vs. Serious/Fatal Injury):")
print_evaluation_metrics(y_test, y_pred_slight)
score_slight = cross_val_score(best_knn_slight, X_test, y_test, cv=5)
print_cross_val_scores(score_slight)
```

```
Best number of neighbors for Slight Injury vs. Serious/Fatal Injury
(oversampled): 1
KNN with Oversampling (Slight Injury vs. Serious/Fatal Injury):
Accuracy:  0.8847058823529412
Confusion Matrix:
 [[1717  414]
 [  76 2043]]
Classification Report:
              precision    recall  f1-score   support

           0       0.96      0.81      0.88      2131
           1       0.83      0.96      0.89      2119

    accuracy                           0.88      4250
   macro avg       0.89      0.88      0.88      4250
weighted avg       0.89      0.88      0.88      4250

Cross-validation scores:  [0.78117647 0.75294118 0.76705882 0.76
0.77882353]
Average score:  0.768
```

##Slight/Fatal Injury (1) vs. Serious Injury (0)

[ ]:
```
X_resampled, y_resampled = oversample.fit_resample(X_scaled, y_serious)

#Check distribution
print(y_resampled.value_counts())
```

```
Accident_serious
1    7164
0    7164
Name: count, dtype: int64
```

[ ]:
```
#Split testing data
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled,␣
 ↪test_size=0.3, random_state=42)


#Train model with best number of neighbors
knn_serious = KNeighborsClassifier()
grid_search_serious = GridSearchCV(knn_serious, param_grid, cv=5,␣
 ↪scoring='accuracy')
grid_search_serious.fit(X_train, y_train)
best_knn_serious = grid_search_serious.best_estimator_

#Create predictiom
y_pred_serious = best_knn_serious.predict(X_test)
```

```
print("Best number of neighbors for Slight/Fatal Injury vs. Serious Injury␣
 ↪(oversampled):", grid_search_serious.best_params_['n_neighbors'])
print("KNN with Oversampling (Slight/Fatal Injury vs. Serious Injury):")
print_evaluation_metrics(y_test, y_pred_serious)
score_serious = cross_val_score(best_knn_serious, X_test, y_test, cv=5)
print_cross_val_scores(score_serious)
```

```
Best number of neighbors for Slight/Fatal Injury vs. Serious Injury
(oversampled): 1
KNN with Oversampling (Slight/Fatal Injury vs. Serious Injury):
Accuracy:  0.8941614328913701
Confusion Matrix:
 [[2066   99]
 [ 356 1778]]
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.95      0.90      2165
           1       0.95      0.83      0.89      2134

    accuracy                           0.89      4299
   macro avg       0.90      0.89      0.89      4299
weighted avg       0.90      0.89      0.89      4299

Cross-validation scores:  [0.78023256 0.76162791 0.76627907 0.78488372
0.78230501]
Average score:  0.775065652326935
```

5.1.1 KNN Model Summary

Target: Slight Injury (1) vs. Serious Injury (0) vs. Fatal Injury (2)

Without Oversampling:

Accuracy: 85.71%

Precision Serious Injury: 86%

Recall: 100%

With Oversampling:

Accuracy: 92.88%

Precision Serious Injury: 96%

Recall: 82%

Target: Slight Injury (0) vs. Serious Injury/Fatal Injury (1)

Without Oversampling:

Accuracy: 85.71%

Precision Serious Injury: 0%

Recall: 0%

With Oversampling:

Accuracy: 88.49%

Precision Serious Injury: 83%

Recall: 96%

Target: Slight/Fatal Injury (1) vs. Serious Injury (0)

Without Oversampling:

Accuracy: 86.60%

Precision Serious Injury: 87%

Recall: 100%

With Oversampling:

Accuracy: 89.46%

Precision Serious Injury: 95%

Recall: 83%

[ ]: