<u>Crash Prediction Report</u>

Syed Mujtaba - 21L-5613
Kamran Ishtiaq - 21L-6253
Khurram Imran - 21L-6256

May 12, 2024

Abstract

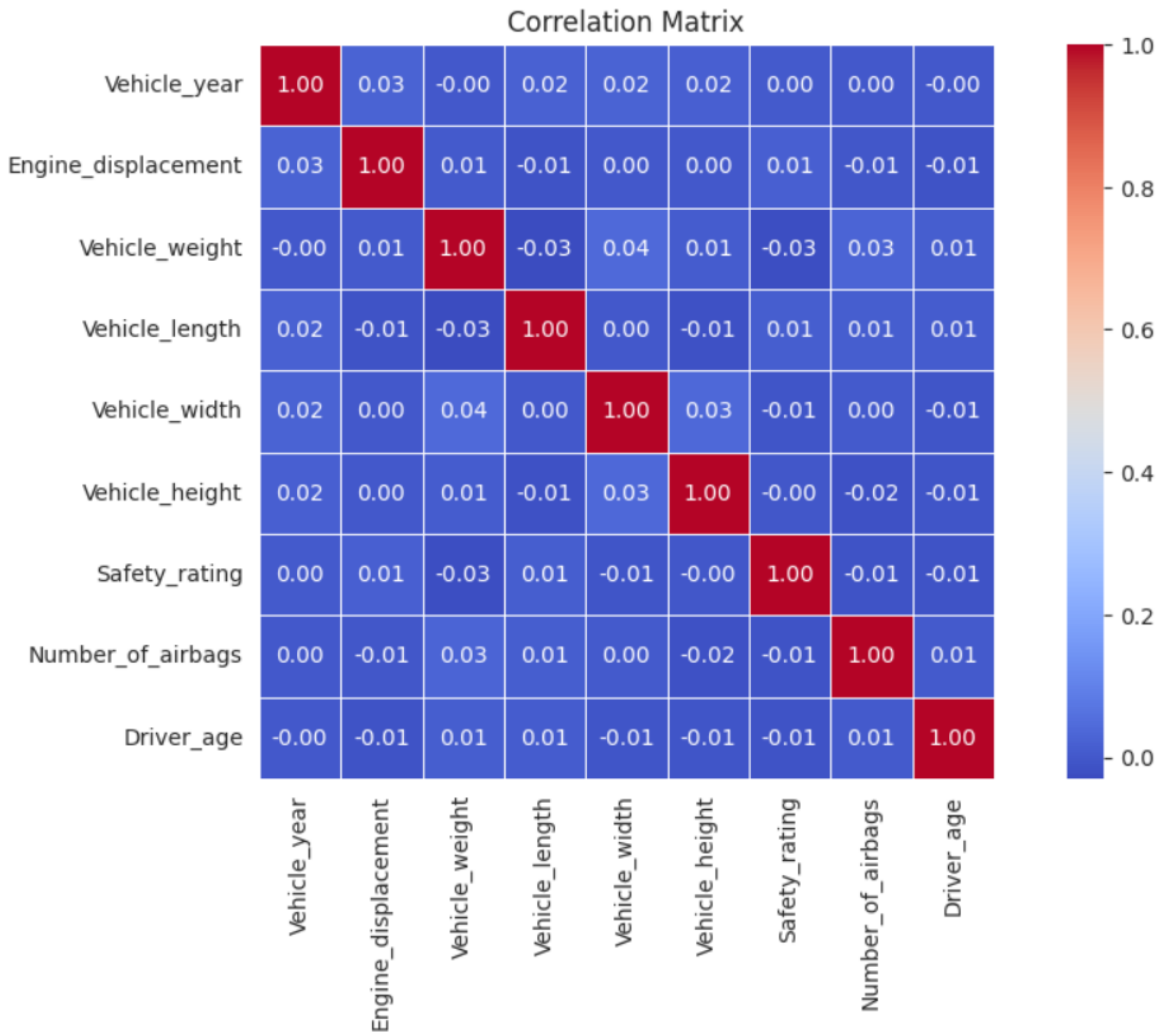
This report summarizes the findings and outcomes of the crash severity classification project conducted using the dataset named ‘crashed data.csv’. The project aimed to predict crash severity based on various attributes related to vehicle crashes using machine learning models and deep learning models.

Introduction

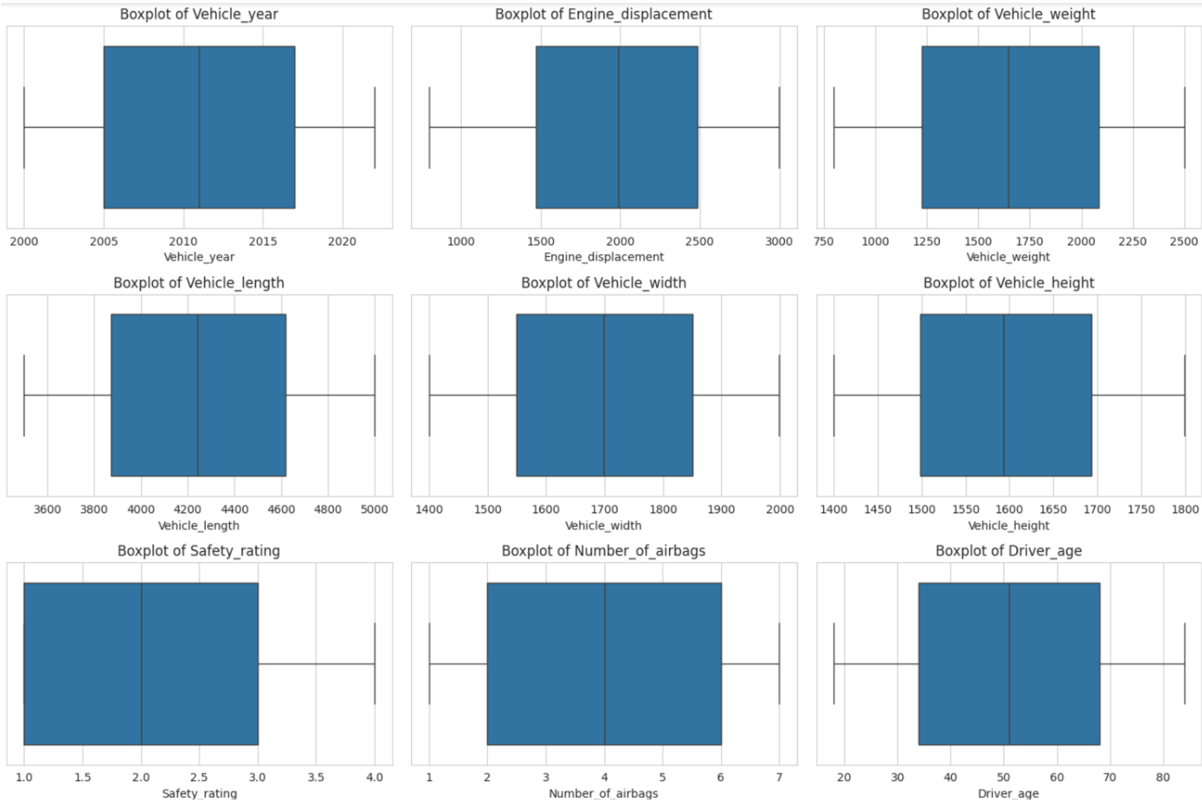
The report aims to provide insights into the crash severity classification project conducted using the dataset named ‘crashed data.csv’. The project focused on analysing various attributes related to vehicle crashes to predict crash severity using machine learning models. This report summarizes the preprocessing steps, model training, and evaluation results.

These are some visualizations of dataset to gain some insights.

HeatMap



Box-Plot of all attributes



These graphs give us some insights of the data present in the dataset and other visualizations are present in code.

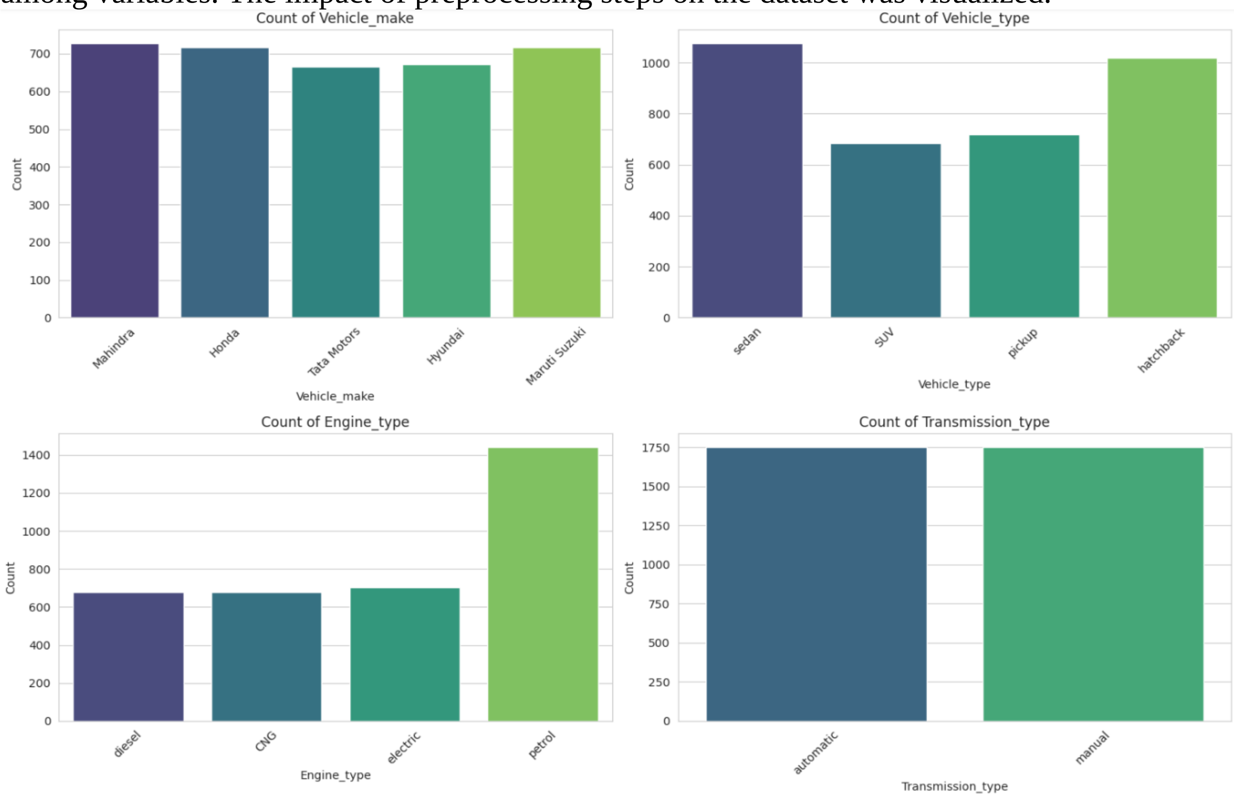
Preprocessing

1. Data Cleaning: Missing values were either removed or imputed using appropriate techniques. Outliers were identified and handled using Z-score or Interquartile Range (IQR) method.

```
# Strategy to handle null values
# For categorical variables, filling null values with the most frequent category (mode)
# For numerical variables, filling null values with the mean or median


# List of categorical columns with missing values
categorical_cols_with_missing = ['ABS_presence', 'ESC_presence', 'TCS_presence', 'TPMS_presence', 'Crash_location', 'Weather_conditions'

# Fill missing values with mode
for col in categorical_cols_with_missing:
    data[col].fillna(data[col].mode()[0], inplace=True)

# List of numeric columns with missing values
numeric_cols_with_missing = []

# Fill missing values with mean or median
for col in numeric_cols_with_missing:
    data[col].fillna(data[col].median(), inplace=True)  # Use median as a measure of central tendency
```

```python
from scipy import stats

# Z-scores for each numeric feature
z_scores = stats.zscore(data[numeric_features])

# Threshold for z-score
threshold = 3

# Find indices of outliers
outlier_indices = (z_scores > threshold).any(axis=1)

# Remove outliers from the DataFrame
data_no_outliers = data[~outlier_indices]

# Plot box plots for numeric features after removing outliers
plt.figure(figsize=(15, 10))

for i, feature in enumerate(numeric_features):
    plt.subplot(3, 3, i+1)
    sns.boxplot(x=data_no_outliers[feature])
    plt.title(f'Boxplot of {feature}')
    plt.xlabel(feature)

plt.tight_layout()
plt.show()
```

2. Data Visualization: Visualizations such as box plots, scatter plots, histograms, and correlation matrices were generated to understand the distribution and relationships among variables. The impact of preprocessing steps on the dataset was visualized.



Model Training

Machine learning models were trained using the pre-processed data. The models trained include:

- Linear Regression

```
# Initialize Linear Regression model
linear_reg = LinearRegression()

# Fit the model on the training data
linear_reg.fit(X_train, y_train)

# Predict on the testing data
y_pred_linear = linear_reg.predict(X_test)

# Evaluate the model
mse_linear = mean_squared_error(y_test, y_pred_linear)
r2_linear = r2_score(y_test, y_pred_linear)

print(f"Mean Squared Error (Linear Regression): {mse_linear}")
print(f"R-squared Score (Linear Regression): {r2_linear}")
```

- Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Initialize Logistic Regression model
logistic_regression_model = LogisticRegression(max_iter=200)

# Train the model
logistic_regression_model.fit(X_train, y_train)

# Make predictions
y_pred = logistic_regression_model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
classification_report = classification_report(y_test, y_pred)

print("Accuracy:", accuracy)
print("Classification Report:")
print(classification_report)
```

- Support Vector Machines (SVM)

```python
import sklearn
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

# Initialize SVM model
svm_model = SVC(kernel='linear', random_state=42)

# Fit the model on the training data
svm_model.fit(X_train, y_train)

# Predict on the testing data
y_pred_svm = svm_model.predict(X_test)

# Evaluate the model
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print(f"Accuracy Score (SVM): {accuracy_svm}")

# Print classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_svm))
```

- Random Forests

```python
# Initialize Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Fit the model on the training data
rf_model.fit(X_train, y_train)

# Predict on the testing data
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Accuracy Score (Random Forest): {accuracy_rf}")

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred_rf))
```

- Naive Bayes

```python
# Initialize Naive Bayes model
nb_model = GaussianNB()

# Fit the model on the training data
nb_model.fit(X_train, y_train)

# Predict on the testing data
y_pred_nb = nb_model.predict(X_test)

# Evaluate the model
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print(f"Accuracy Score (Naive Bayes): {accuracy_nb}")

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred_nb))
```

- K-Nearest Neighbours (KNN)

```python
# Initialize KNN model
knn_model = KNeighborsClassifier(n_neighbors=5)

# Fit the model on the training data
knn_model.fit(X_train, y_train)

# Predict on the testing data
y_pred_knn = knn_model.predict(X_test)

# Evaluate the model
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f"Accuracy Score (KNN): {accuracy_knn}")

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred_knn))
```

- Decision Trees

```python
# Initialize Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)

# Fit the model on the training data
dt_model.fit(X_train, y_train)

# Predict on the testing data
y_pred_dt = dt_model.predict(X_test)

# Evaluate the model
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"Accuracy Score (Decision Tree): {accuracy_dt}")

# Classification Report
print("Classification Report:")
print(classification_report(y_test, y_pred_dt))
```

- ANN

```python
# Initialize ANN model
ann_model = Sequential()

# Add input layer
ann_model.add(Dense(128, input_dim=X_train_scaled.shape[1], activation='relu'))
ann_model.add(Dropout(0.2))  # Dropout layer to prevent overfitting

# Add hidden layer
ann_model.add(Dense(64, activation='relu'))
ann_model.add(Dropout(0.2))

# Add output layer
ann_model.add(Dense(1, activation='sigmoid'))  # Sigmoid activation for binary classification
```

```python
# Compile the model
ann_model.compile(loss='binary_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])
```

```python
# Train the model
history = ann_model.fit(X_train_scaled, y_train, epochs=10, batch_size=32, validation_split=0.2, verbose=1)
```

Output Prediction

Binary classification was performed to predict crash severity (severe or moderate) using the trained models. The predictions for each model were outputted, and their performance was evaluated.

Model Evaluation Results

1. Linear Regression: Achieved an accuracy of 99.18%.

```
Mean Squared Error (Linear Regression): 0.0014313037169374962
R-squared Score (Linear Regression): 0.9918102337650126
```

2. Logistic Regression: Achieved an accuracy of 98.28%.

```
Accuracy: 0.9828571428571429
Classification Report:
              precision    recall  f1-score   support

       False       0.98      0.94      0.96       158
        True       0.98      0.99      0.99       542

    accuracy                           0.98       700
   macro avg       0.98      0.97      0.98       700
weighted avg       0.98      0.98      0.98       700
```

3. Support Vector Machines (SVM): Achieved an accuracy of 99.71%.

```
Accuracy Score (SVM): 0.9971428571428571
Classification Report:
              precision    recall  f1-score   support

       False       1.00      0.99      0.99       158
        True       1.00      1.00      1.00       542

    accuracy                           1.00       700
   macro avg       1.00      0.99      1.00       700
weighted avg       1.00      1.00      1.00       700
```

4. Random Forests: Achieved an accuracy of 99.85%.

```
Accuracy Score (Random Forest): 0.9985
Classification Report:
              precision    recall  f1-score   support

       False       1.00      0.99      1.00       400
        True       1.00      1.00      1.00      1600

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

5. Naive Bayes: Achieved an accuracy of 99.5%.

```
Accuracy Score (Naive Bayes): 0.995
Classification Report:
              precision    recall  f1-score   support

       False       0.98      0.99      0.99       400
        True       1.00      1.00      1.00      1600

    accuracy                           0.99      2000
   macro avg       0.99      0.99      0.99      2000
weighted avg       1.00      0.99      1.00      2000
```

6. K-Nearest Neighbours (KNN): Achieved an accuracy of 76.65%.

```
Accuracy Score (KNN): 0.7665
Classification Report:
              precision    recall  f1-score   support

       False       0.28      0.11      0.16       400
        True       0.81      0.93      0.86      1600

    accuracy                           0.77      2000
   macro avg       0.55      0.52      0.51      2000
weighted avg       0.70      0.77      0.72      2000
```

7. Decision Trees: Achieved an accuracy of 99.75%.

```
Accuracy Score (Decision Tree): 0.9975
Classification Report:
              precision    recall  f1-score   support

       False       0.99      0.99      0.99       400
        True       1.00      1.00      1.00      1600

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

8. Artificial Neural Network (ANN): Achieved an accuracy of 99.85%.

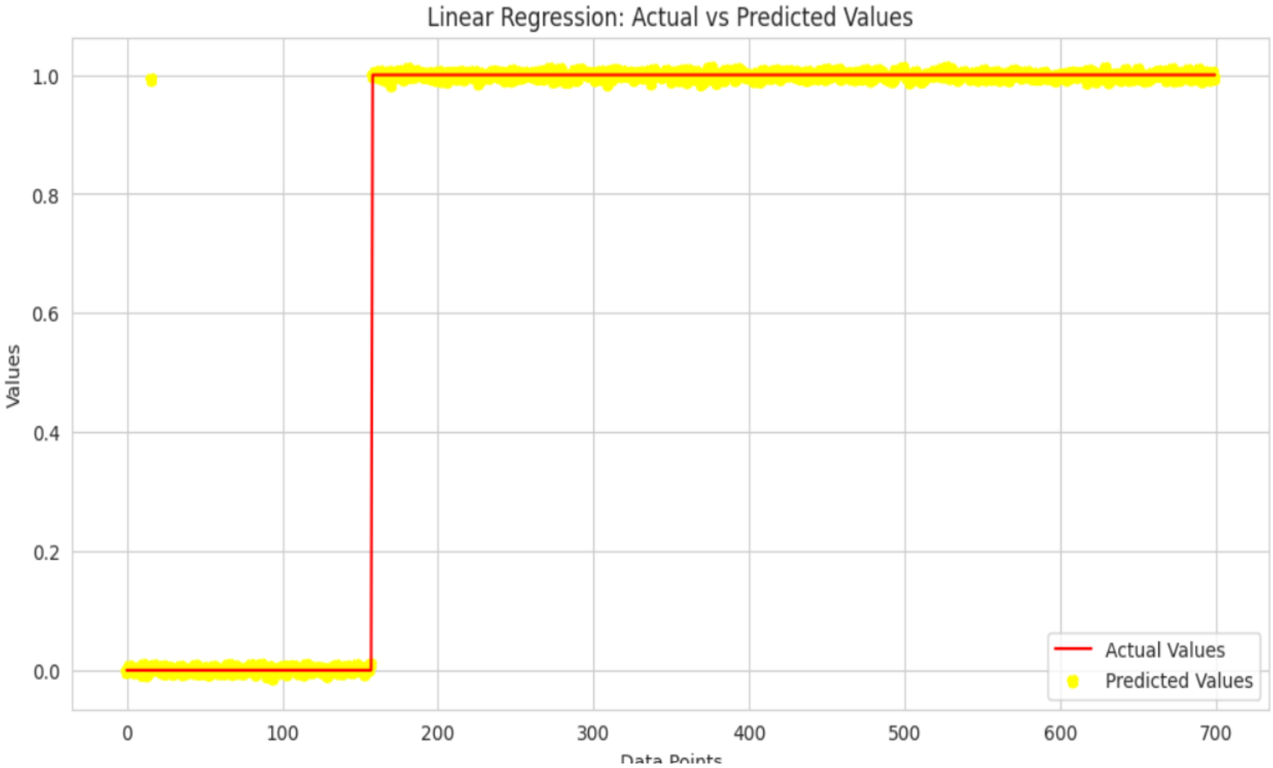
```
Accuracy Score (ANN): 0.9985
Classification Report:
              precision    recall  f1-score   support

       False       1.00      0.99      1.00       400
        True       1.00      1.00      1.00      1600

    accuracy                           1.00      2000
   macro avg       1.00      1.00      1.00      2000
weighted avg       1.00      1.00      1.00      2000
```

Discussion

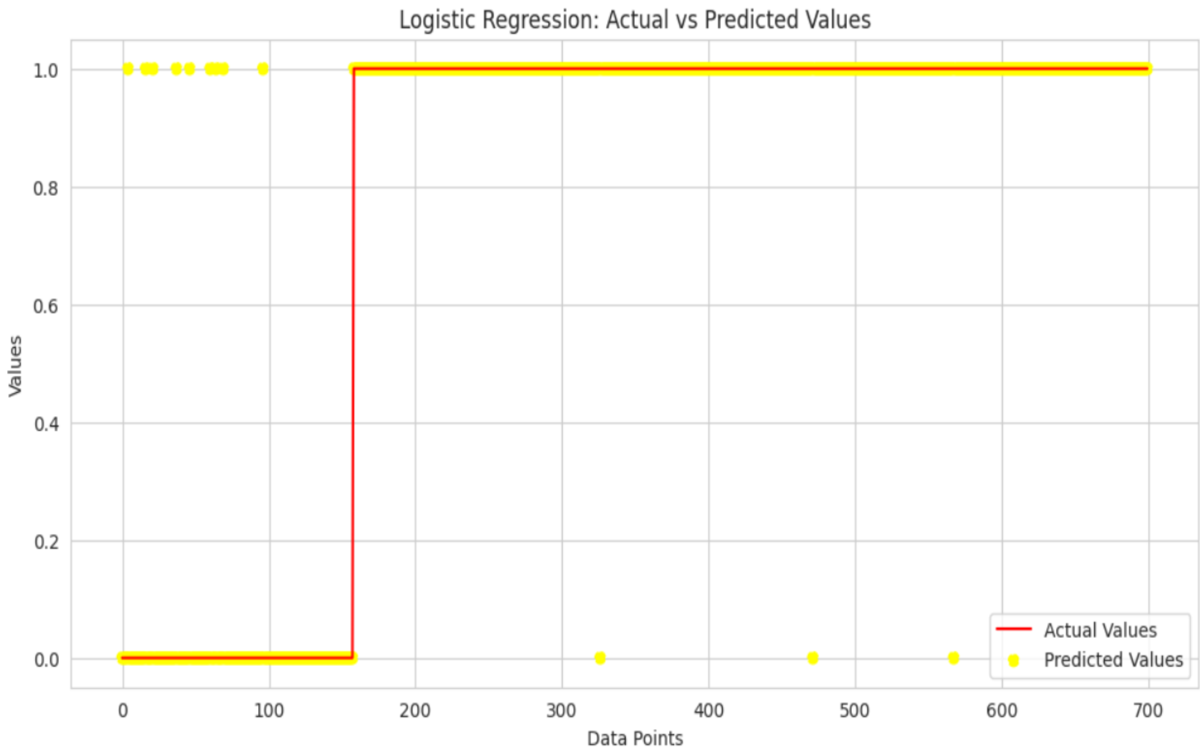
Most of the models achieved high accuracies, indicating their effectiveness in predicting crash severity. However, K-Nearest Neighbours (KNN) showed comparatively lower accuracy. This could be due to the lazy learning nature of KNN and the dataset's characteristics. Logistic Regression also exhibited slightly lower accuracy compared to other models, which could be attributed to its linear nature and the complexity of the dataset.

Visualizations

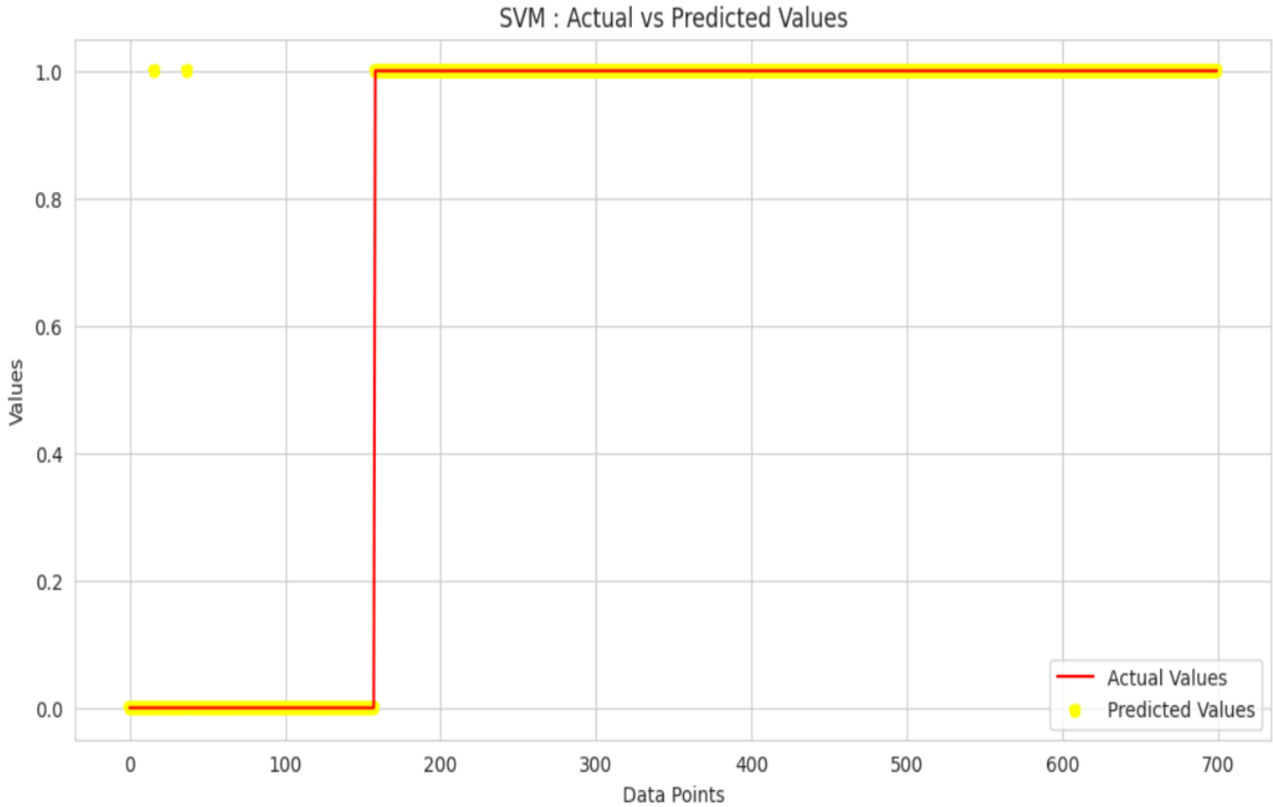
1. **Linear Regression**

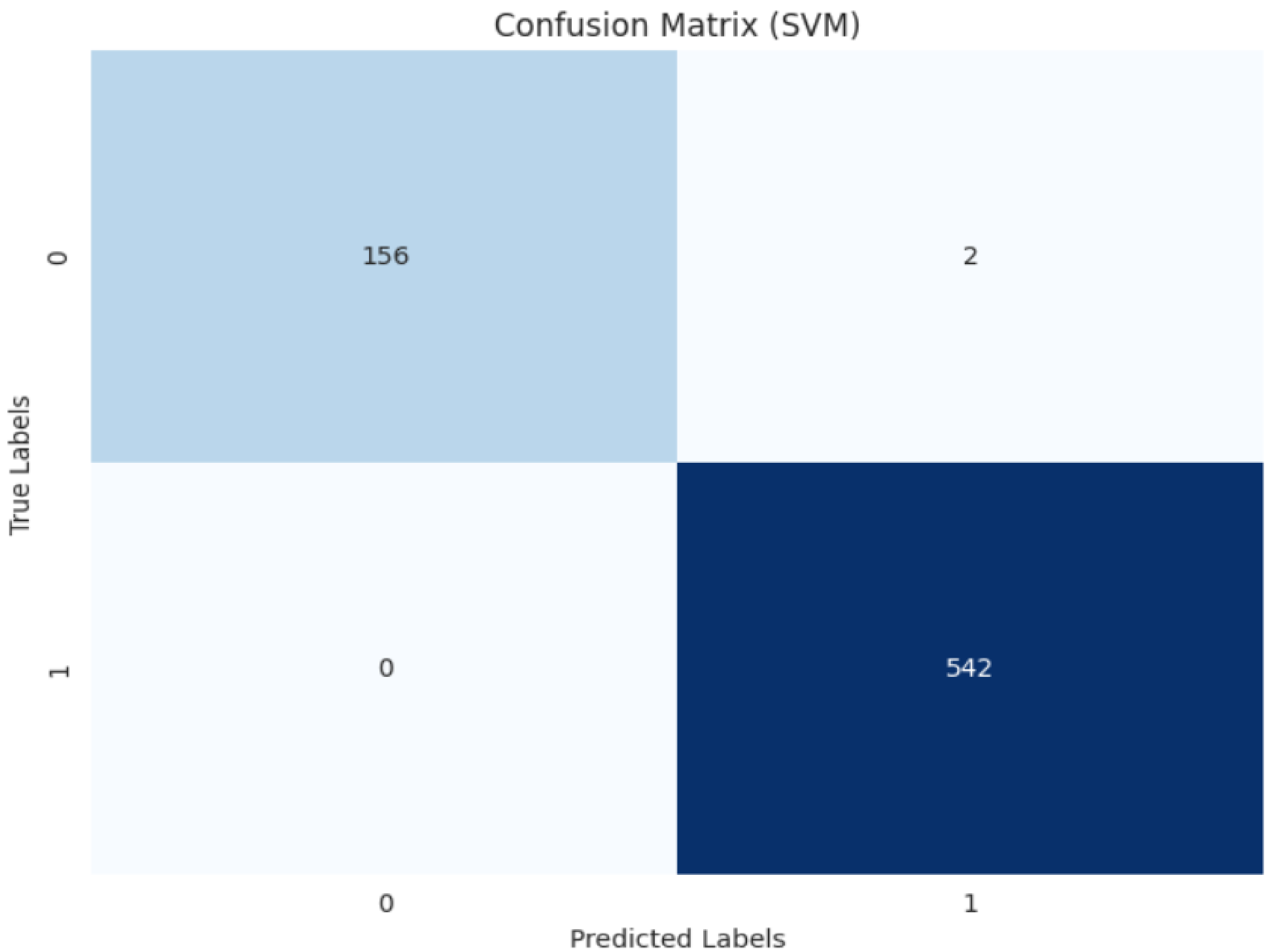


2. **Logistic Regression**

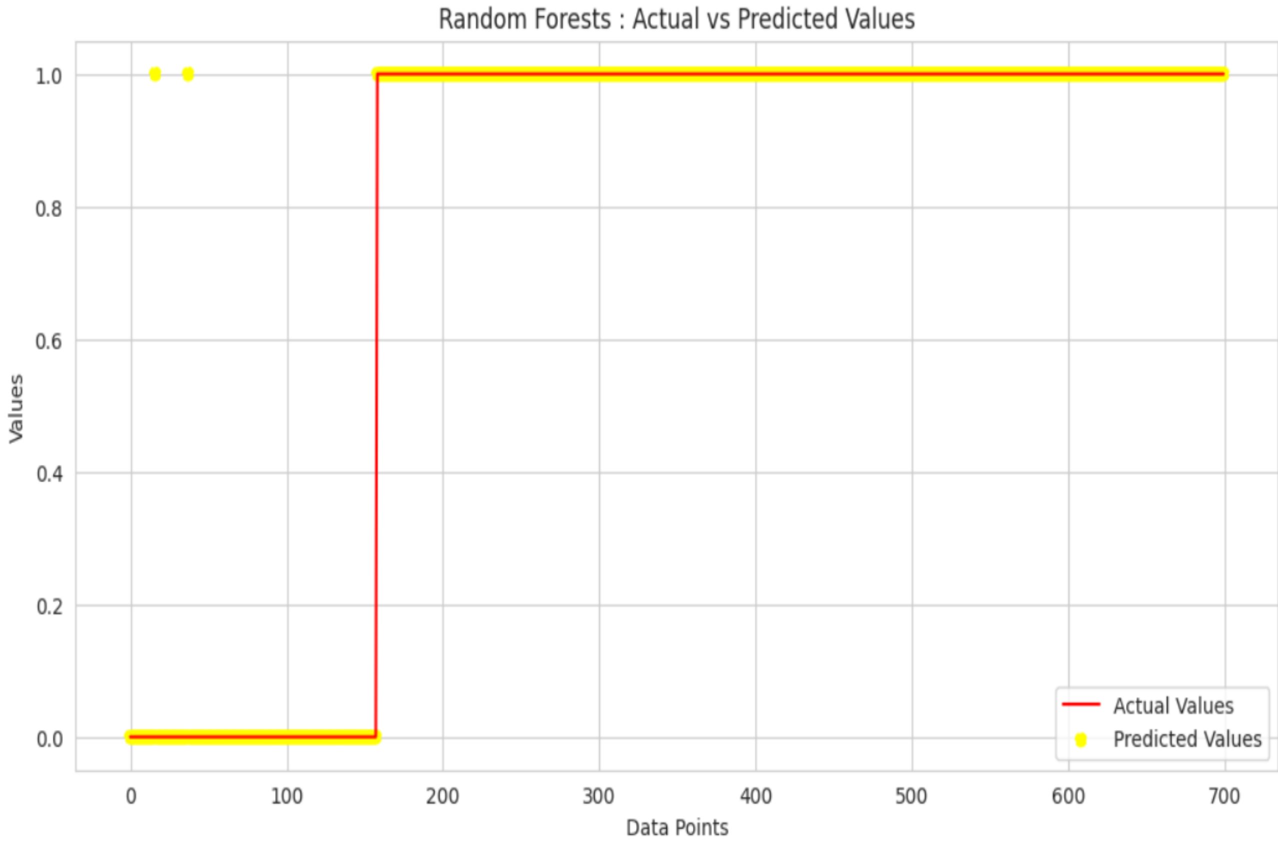


3. Support Vector Machines (SVM)

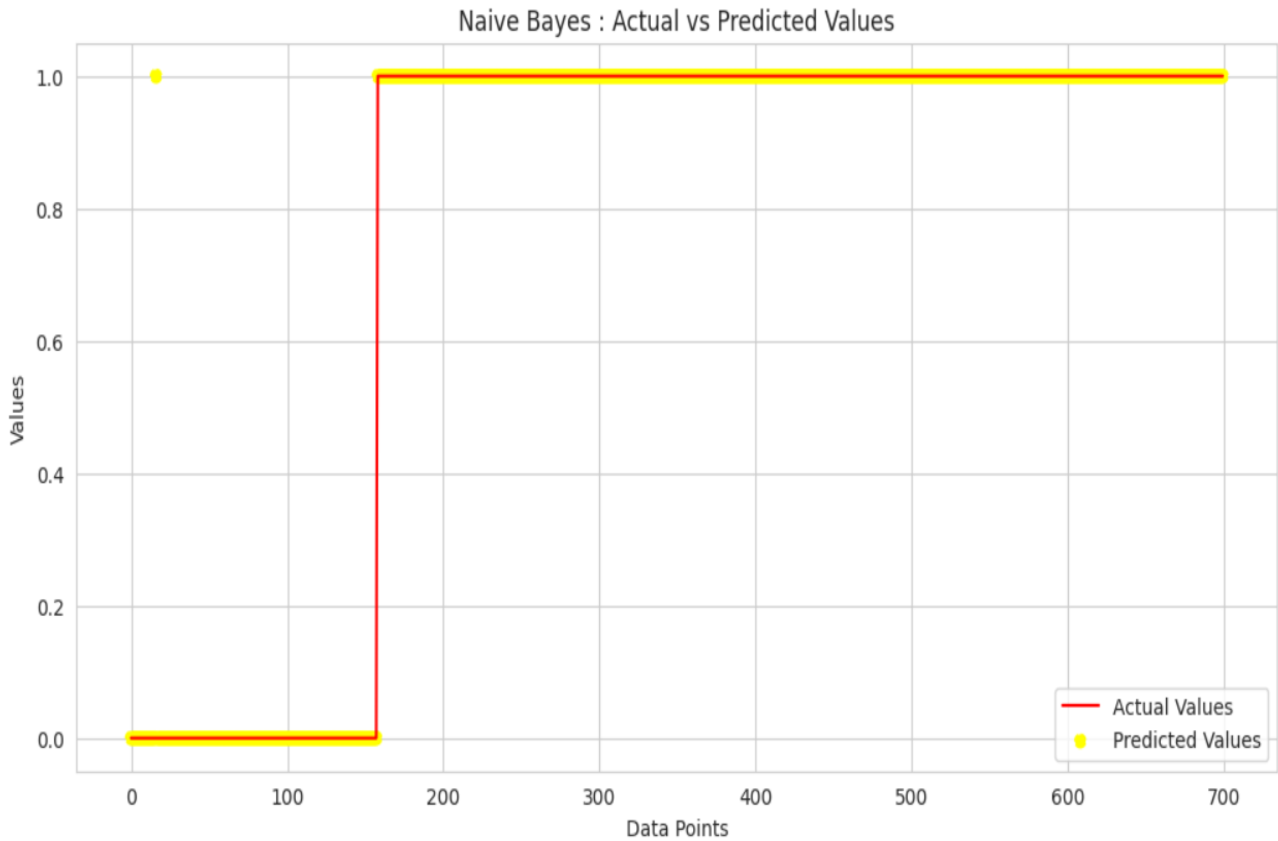




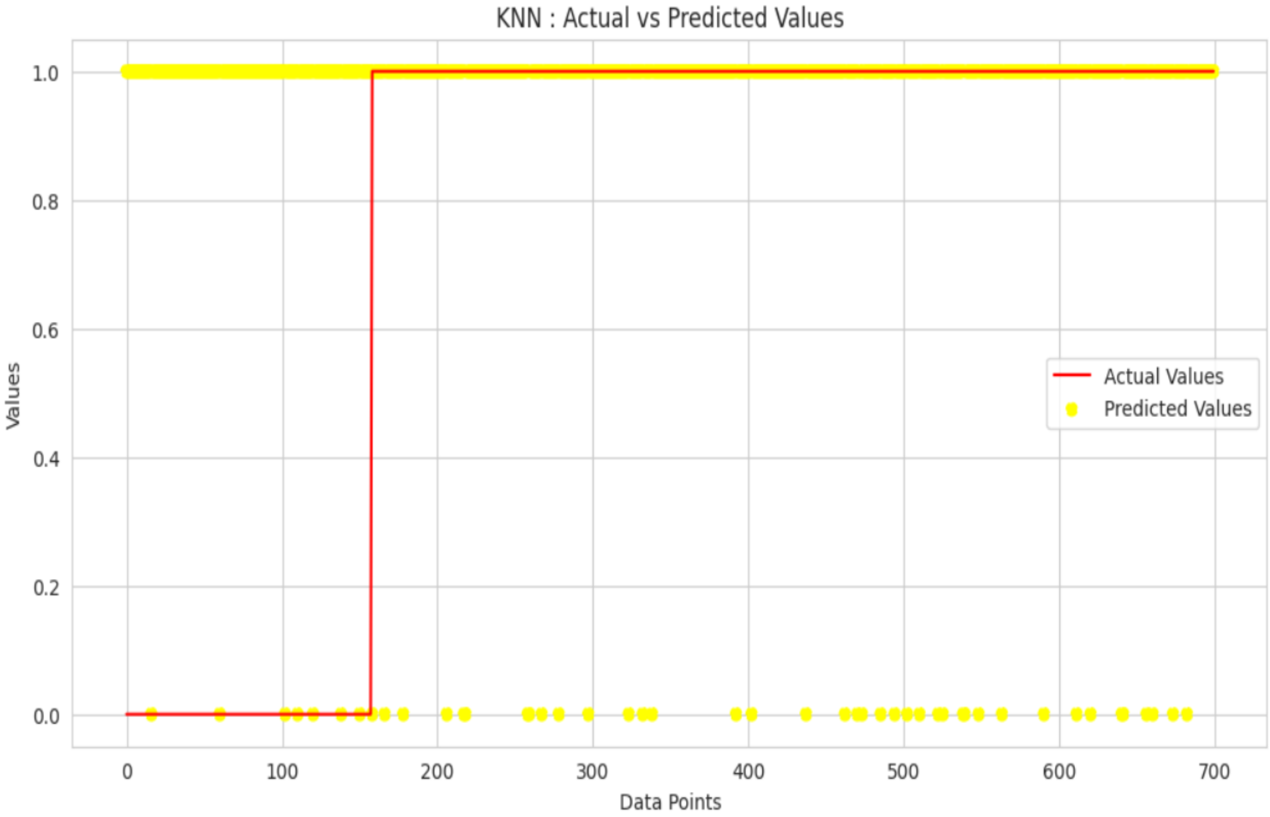
4. Random Forests



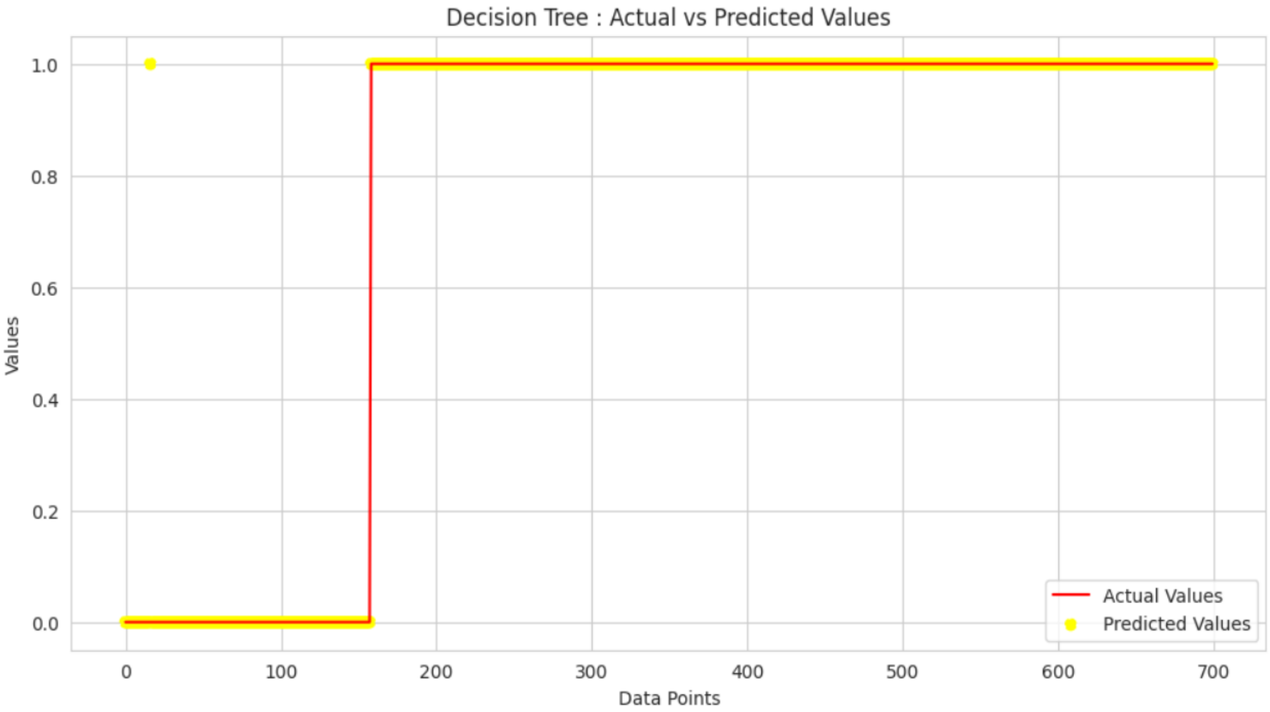
5. Naive Bayes

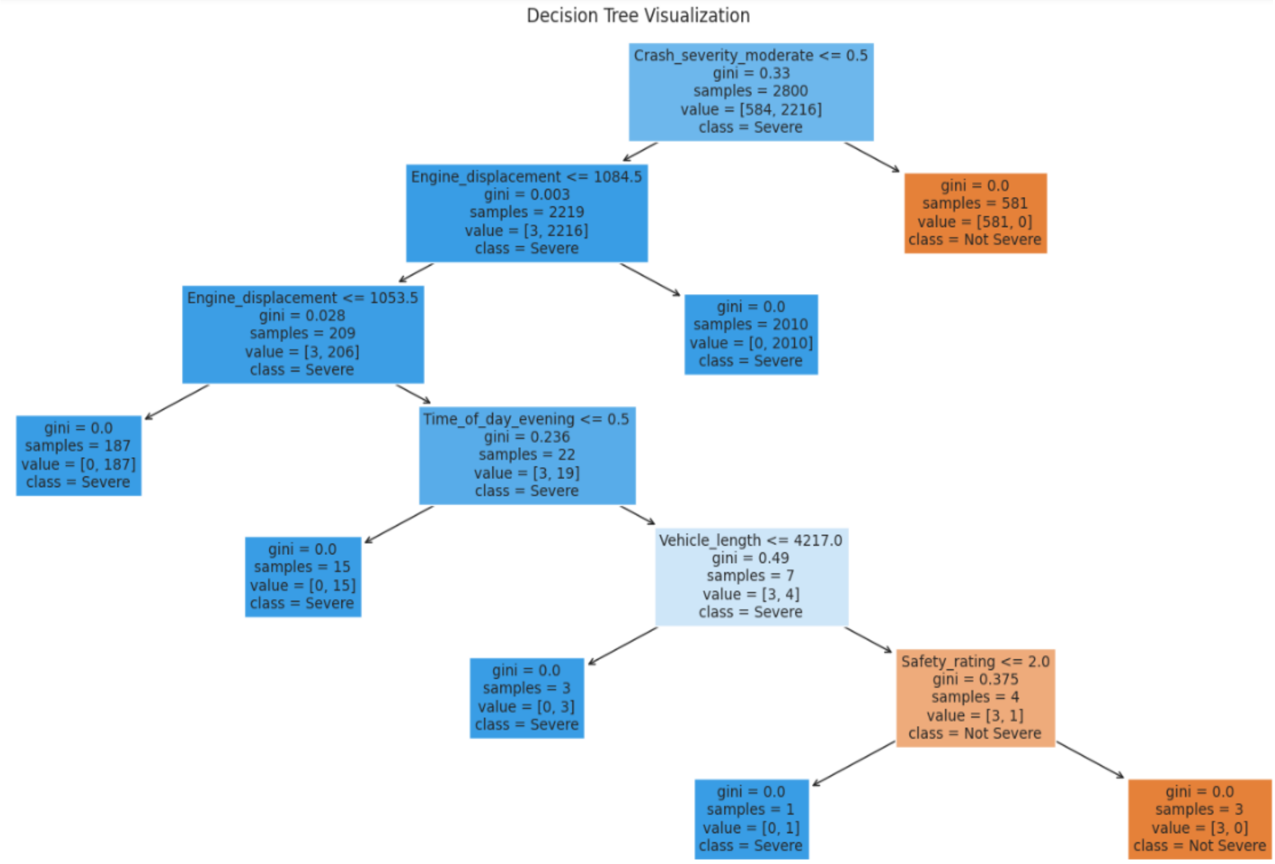


6. K-Nearest Neighbours (KNN)

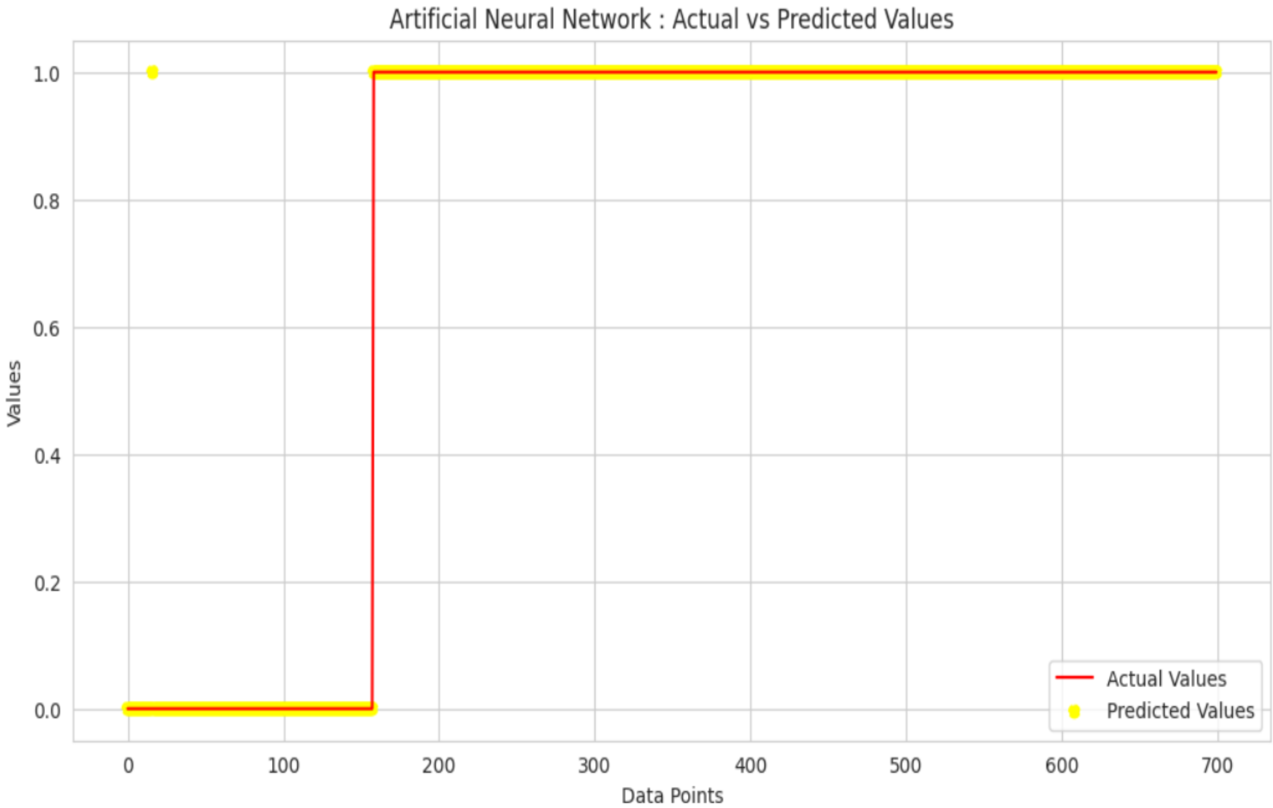


7. Decision Trees

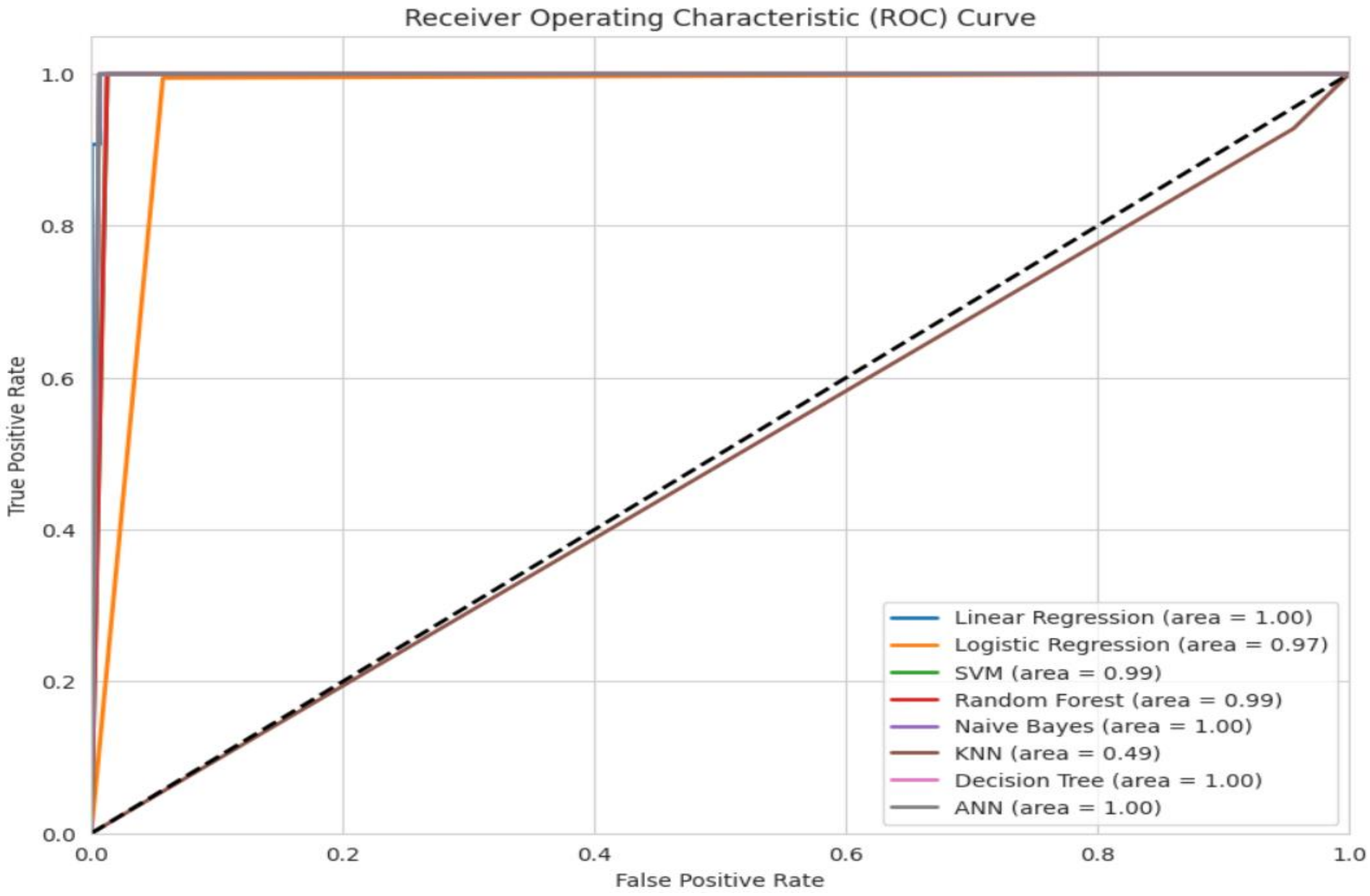




8. Artificial Neural Network (ANN)



ROC-Curve of all Models



Predicting Accident Severity with ALL Models

```
# Define a single input data point
input_data = {
    'Vehicle_make': 'Honda',
    'Vehicle_type': 'sedan',
    'Vehicle_year': 2014,
    'Engine_type': 'CNG',
    'Engine_displacement': 2959,
    'Transmission_type': 'manual',
    'Number_of_cylinders': 4,
    'Vehicle_weight': 1949,
    'Vehicle_length': 3947,
    'Vehicle_width': 1933,
    'Vehicle_height': 1719,
    'Safety_rating': 1,
    'Number_of_airbags': 5,
    'ABS_presence': 0,
    'ESC_presence': 1,
    'TCS_presence': 0,
    'TPMS_presence': 1,
    'Crash_location': 'urban',
    'Weather_conditions': 'clear',
    'Road_surface_conditions': 'dry',
    'Time_of_day': 'night',
    'Day_of_week': 'Saturday',
    'Driver_age': 80,
    'Driver_gender': 'Female',
    'Vehicle_age': 10,
    'Driver_experience': 62
}

# Creating a DataFrame with the single input data point
input_df = pd.DataFrame([input_data])
```

The actual output of this input data is **“severe”**.

```
# Applying one-hot encoding to categorical variables
categorical_features = ['Vehicle_make', 'Vehicle_type', 'Engine_type', 'Transmission_type',
                        'Crash_location', 'Weather_conditions', 'Road_surface_conditions',
                        'Time_of_day', 'Day_of_week', 'Driver_gender']
input_encoded = pd.get_dummies(input_df, columns=categorical_features)

# Load your trained model and scaler
# Assuming you have X_train, y_train, and scaler loaded from your previous training

# Ensuring the same column order as in the training data
# Assuming X_train.columns holds the column names
input_encoded = input_encoded.reindex(columns=X_train.columns, fill_value=0)

# Scaling numerical variables using the same scaler used for training
input_scaled = scaler.transform(input_encoded)
```

1. Prediction using Linear Regression

```
# Linear Regression

linear_reg = LinearRegression()

# Train
linear_reg.fit(X_train, y_train)

# Predict the severity
linear_predicted_class_label = linear_reg.predict(input_scaled)

linear_predicted_class_label = 'severe' if linear_predicted_class_label[0] else 'moderate'
print("Predicted Class:", linear_predicted_class_label)
print()
print()
```

```
Predicted Class: severe
```

2. Prediction using Logistic Regression

```
# Logisitic Regression

logistic_reg = LogisticRegression()

# Train model
logistic_reg.fit(X_train, y_train)

# Predict the class (severe or moderate)
predicted_class = logistic_reg.predict(input_scaled)
predicted_class_label = 'severe' if predicted_class[0] else 'moderate'
print("Predicted Class:", predicted_class_label)
print()
print()
```

```
Predicted Class: severe
```

3. Prediction using Support Vector Machine (SVM)

```
# SVM

svm_model = SVC()

# Train model
svm_model.fit(X_train, y_train)

# Predict the class
svm_predicted_class = svm_model.predict(input_scaled)

svm_predicted_class_label = 'severe' if svm_predicted_class[0] else 'moderate'

print("Predicted Class (SVM):", svm_predicted_class_label)
print()
print()
```

```
Predicted Class (SVM): severe
```

4. Prediction using Random Forest

```
# Random Forest

rf_model = RandomForestClassifier()

# Train model
rf_model.fit(X_train, y_train)

# Predict the class
rf_predicted_class = rf_model.predict(input_scaled)

# Map the predicted class to its label
rf_predicted_class_label = 'severe' if rf_predicted_class[0] else 'moderate'

print("Predicted Class (Random Forest):", rf_predicted_class_label)

print()
print()
```

```
Predicted Class (Random Forest): severe
```

5. Prediction using Naive Bayes

```
# Naive Bayes model

nb_model = GaussianNB()

# Train model
nb_model.fit(X_train, y_train)

# Predict the class
nb_predicted_class = nb_model.predict(input_scaled)

# Map the predicted class to its label
nb_predicted_class_label = 'severe' if nb_predicted_class[0] else 'moderate'

print("Predicted Class (Naive Bayes):", nb_predicted_class_label)

print()
print()
```

```
Predicted Class (Naive Bayes): severe
```

6. Prediction using K-Nearest Neighbour (KNN)

```
# KNN

knn_model = KNeighborsClassifier()

# Train model
knn_model.fit(X_train, y_train)

# Predict the class
knn_predicted_class = knn_model.predict(input_scaled)

# Map the predicted class to its label
knn_predicted_class_label = 'severe' if knn_predicted_class[0] else 'moderate'

print("Predicted Class (KNN):", knn_predicted_class_label)

print()
print()
```

```
Predicted Class (KNN): moderate
```

7. Prediction using Decision Tree

```
# Decision Tree model
dt_model = DecisionTreeClassifier()

# Train your model
dt_model.fit(X_train, y_train)

# Predict the class
dt_predicted_class = dt_model.predict(input_scaled)

# Map the predicted class to its label
dt_predicted_class_label = 'severe' if dt_predicted_class[0] else 'moderate'

print("Predicted Class (Decision Tree):", dt_predicted_class_label)
print()
print()
```

```
Predicted Class (Decision Tree): severe
```

8. Prediction using Artificial Neural Network (ANN)

```
# ANN
ann_model = ann_model = Sequential()

# Assuming your ANN model is already defined and compiled

# Compile the model
ann_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Predict the class probabilities
ann_predicted_probabilities = ann_model.predict(input_scaled)

# Get the predicted class index
ann_predicted_class_index = tf.argmax(ann_predicted_probabilities, axis=1)

# Map the predicted class index to its label
ann_predicted_class_label = 'severe' if ann_predicted_class_index[0] == 0 else 'moderate'
print()
print("Predicted Class (ANN):", ann_predicted_class_label)

print()
print()
```

```
WARNING:tensorflow:6 out of the last 6 calls to <function Model.make_predict_function.<locals>.pre
1/1 [==============================] - 0s 456ms/step

Predicted Class (ANN): severe
```

Conclusion

The project successfully developed machine learning models to classify crash severity based on vehicle characteristics, environmental conditions, and driver demographics. The models exhibited high accuracies, except for K-Nearest Neighbours (KNN). Further optimization and fine-tuning of models may enhance their performance. These predictive models can contribute to the development of effective safety strategies and interventions in reducing crash severity.

Recommendations

- Further analysis can be conducted to understand the reasons behind KNN's lower accuracy and explore methods to improve its performance.

- Feature engineering techniques can be applied to extract more meaningful information from the dataset, potentially enhancing the models' predictive capabilities.

- Ensemble learning methods can be explored to combine the strengths of multiple models and improve overall performance.

References

The dataset used for this analysis is available on Kaggle:

https://www.kaggle.com/datasets/swish9/synthetic-indian-automobile-crash-data

This report summarizes the key findings and outcomes of the crash severity classification project, providing valuable insights for further analysis and improvement.