ATeam06_Crash_Severity_Prediction_Model

December 6, 2023

Colab File Link: https://colab.research.google.com/drive/13OscFy7FsFjkcQw1KFyFBMN9bCbw-tcY?usp=sharing

1 Crash Severity Prediction Model

Submitted by: 1. Megha Arul Senthilkumar 2. Neeharika Kamireddy 3. Rajashree Ramaprabu

1.1 1. Introduction

1.1.1 *Problem Definition*

The objective of this project is to develop a machine learning model capable of predicting the severity of driver injuries and the extent of vehicle damage in the aftermath of a collision. The model will take into account various input variables, including collision type, weather conditions, traffic density, light conditions, and the presence of substance abuse. The goal is to analyze the impact of these input features on the respective target variables (Injury Severity and Vehicle Damage Extent) and create a classification model that can effectively assess the severity of a collision (in terms of driver's injury and vehicle damage) based on the given circumstances.

Background: In the realm of road safety and accident prevention, the ability to accurately predict the severity of driver injuries and the extent of vehicle damage plays a pivotal role. Understanding the intricate relationship between various factors is essential for developing effective preventive measures. By harnessing the power of machine learning, we aim to create a robust detection model that not only considers the individual impact of these factors but also comprehensively analyzes their combined effect on the severity of driver injuries and the damage sustained by the vehicles involved. This model holds the potential to revolutionize how we approach road safety, providing valuable insights into the circumstances that contribute to different outcomes in collisions.

Challenges: The challenges in developing such a detection model are multifaceted. First and foremost, the complexity of real-world collisions introduces a wide array of variables that must be carefully considered. Factors such as unanticipated road conditions, human behaviors and situations, and diverse vehicle types add layers of intricacy to the modeling process. Overcoming these challenges requires a meticulous approach to data preprocessing, feature engineering, and model optimization to ensure the reliability and robustness of the final detection model.

Machine Learning in Action: Machine learning comes into play as a powerful tool to discern patterns and relationships within the vast dataset of collision records. Employing classification techniques, such as decision trees, logistic regression, and ensemble methods like random forests, allows us to build a predictive model that learns from historical data. The model leverages the input feature to make informed predictions about the severity of driver injuries and the extent of vehicle damage. By continuously refining its understanding through iterative training, the model adapts to the intricacies of real-world scenarios, making it a dynamic and valuable tool for proactive decision-making in the realm of road safety.

Outcome: The anticipated outcome of this machine learning initiative is the creation of a sophisticated detection model capable of accurately predicting driver injury severity and vehicle damage extent in collision scenarios. Beyond its predictive capabilities, the model has the potential to significantly enhance our understanding of the complex factors contributing to various outcomes in road incidents. Stakeholders, including law enforcement agencies and policymakers, could leverage these insights to implement targeted interventions, leading to optimized resource allocation, more efficient emergency response, and improved traffic management. The model's impact extends to areas such as insurance risk assessment, public awareness campaigns, legal considerations, and the development of innovative safety technologies. Ultimately, the deployment of this predictive model stands to contribute comprehensively to the overarching goal of reducing the frequency and severity of road accidents, potentially saving lives and mitigating injuries in the realm of road safety.

Task: Predict the driver's injury severity and the extent of vehicle damage, using classification, given a huge data of vehicle crashes and their features.

1.2 2. Dataset Source and Overview

1.2.1 2.1 Data Source

This is a dataset from the US government site (data.gov) which contains information about vehicle crash reporting. Link here.

This dataset provides information on motor vehicle traffic collisions. The dataset reports details of all traffic collisions occurring on county and local roadways within Montgomery County, as collected via the Automated Crash Reporting System (ACRS) of the Maryland State Police, and reported by the Montgomery County Police, Gaithersburg Police, Rockville Police, or the Maryland-National Capital Park Police. This dataset shows each collision data recorded and the drivers involved.

1.2.2 2.2 Dataset Description

This dataset contains one table named, Crash_Reporting_Drivers_Data, which consists of 168850 rows and 43 columns.

1. Report Number: ACRS Report Number assigned to the incident.
2. Local Case Number: Case number from the local investigating agency for the incident.
3. Agency Name: Name of the investigating agency
4. ACRS Report Type: Identifies crash as property, injury or fatal
5. Crash Date/Time: Date and Time of crash
6. Route Type: Type of roadway at crash location
7. Road Name: Name of road
8. Cross-Street Type: Roadway type for nearest cross-street

9. Cross-Street Name: Name of nearest cross-street
10. Off-Road Description: Description of location for off-road collisions.
11. Municipality: Jurisdiction for crash location
12. Related Non-Motorist: Type(s) of Non-motorist involved
13. Collision Type: Type of collision
14. Weather: Weather at collision location
15. Surface Condition: Condition of roadway surface
16. Light: Lighting conditions
17. Traffic Control: Signage or traffic control devices
18. Driver Substance Abuse: Substance abuse detected for all drivers involved
19. Non-Motorist Substance Abuse: Substance abuse detected for all non-motorists involved
20. Person ID: Unique identifier for non-motorist
21. Driver at Fault: Whether the driver was at fault
22. Injury Severity: Severity of injury to the driver
23. Circumstance: Circumstance(s) specific to this driver.
24. Driver Distracted By: The reason the driver was distracted
25. Drivers License State: The state the driver's license was issued
26. Vehicle ID: The unique identifier for the driver's vehicle.
27. Vehicle Damage Extent: The severity of the vehicle damage
28. Vehicle First Impact Location: Vehicle - Location of vehicle area where first impact occurred on.
29. Vehicle Second Impact Location: Vehicle - Location of vehicle area where second impact occurred on.
30. Vehicle Body Type: The body type of the vehicle
31. Vehicle Movement: The movement of the vehicle at the time of collision
32. Vehicle Continuing Dir: Vehicle Circumstances - Continuation direction of vehicle after collisions
33. Vehicle Going Dir: Vehicle Circumstances - Movement of vehicle before collision
34. Speed Limit: Vehicle Circumstances - Local Area posted speed limit
35. Driverless Vehicle: Vehicle Circumstances - If the vehicle was driverless or not
36. Parked Vehicle: Vehicle - Defines if the vehicle was parked or not at the event.
37. Vehicle Year: The vehicle's year
38. Vehicle Make: Make of the vehicle
39. Vehicle Model: Model of the vehicle
40. Equipment Problems: Driver - Improper use of safety equipment issues
41. Latitude: Y coordinate of crash location
42. Longitude: X coordinate of crash location
43. Location: Location of crash

1.3 3. Data Loading and Cleaning

1.3.1 3.1 Import Packages and Libraries

[ ]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
```

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer, make_column_selector
from sklearn import set_config
from scipy.stats import chi2_contingency
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score,␣
↪balanced_accuracy_score
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.preprocessing import LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.ensemble import StackingClassifier
from sklearn.pipeline import make_pipeline
from sklearn.tree import plot_tree
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import uniform
from sklearn.feature_selection import RFECV
from scipy.stats import randint, loguniform
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.neighbors import KNeighborsClassifier
from mlxtend.feature_selection import SequentialFeatureSelector
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import VotingClassifier
```

1.3.2 3.2 Loading Data

Mount Google Drive

[ ]:
```
#from google.colab import drive
#drive.mount('/content/drive')
```

[ ]:
```
data = pd.read_csv('Crash_Reporting_-_Drivers_Data.csv')
data.head(5)
```

```
/var/tmp/ipykernel_6156/1074682724.py:1: DtypeWarning: Columns (1) have mixed
types. Specify dtype option on import or set low_memory=False.
  data = pd.read_csv('Crash_Reporting_-_Drivers_Data.csv')
```

[ ]:
```
  Report Number Local Case Number                 Agency Name  \
0   MCP3040003N         190026050   Montgomery County Police
1    EJ78850038         230034791  Gaithersburg Police Depar
2   MCP2009002G         230034583   Montgomery County Police
3   MCP3201004C         230035036   Montgomery County Police
```

```
4   MCP23290028         230035152   Montgomery County Police

       ACRS Report Type         Crash Date/Time        Route Type  \
0  Property Damage Crash  05/31/2019 03:00:00 PM               NaN
1  Property Damage Crash  07/21/2023 05:59:00 PM  Maryland (State)
2  Property Damage Crash  07/20/2023 03:10:00 PM  Maryland (State)
3  Property Damage Crash  07/23/2023 12:10:00 PM            County
4  Property Damage Crash  07/24/2023 06:10:00 AM            County

               Road Name Cross-Street Type  Cross-Street Name  \
0                    NaN               NaN                NaN
1           FREDERICK RD           Unknown    WATKINS MILL RD
2            GEORGIA AVE  Maryland (State)         NORBECK RD
3        CRYSTAL ROCK DR            County  WATERS LANDING DR
4  MONTGOMERY VILLAGE AVE            County       CENTERWAY RD

            Off-Road Description  … Speed Limit Driverless Vehicle  \
0  PARKING LOT OF 3215 SPARTAN RD  …          15                 No
1                             NaN  …          40                 No
2                             NaN  …          35                 No
3                             NaN  …          40                 No
4                             NaN  …          35                 No

  Parked Vehicle Vehicle Year Vehicle Make Vehicle Model Equipment Problems  \
0             No         2004        HONDA            TK            UNKNOWN
1             No         2011          GMC            TK         NO MISUSE
2             No         2019         FORD          F150         NO MISUSE
3             No         2016          KIA            SW         NO MISUSE
4             No         2016         TOYT            TK         NO MISUSE

    Latitude  Longitude                     Location
0  39.150044 -77.063089  (39.15004368, -77.06308884)
1  39.159264 -77.219025   (39.1592635, -77.21902483)
2  39.109535 -77.075806  (39.10953506, -77.07580619)
3  39.190149 -77.266766  (39.19014917, -77.26676583)
4  39.172558 -77.203745  (39.17255801, -77.20374546)

[5 rows x 43 columns]
```

[ ]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169760 entries, 0 to 169759
Data columns (total 43 columns):
 #   Column                         Non-Null Count   Dtype
---  ------                         --------------   -----
 0   Report Number                  169760 non-null  object
```

```
 1   Local Case Number              169760 non-null  object
 2   Agency Name                    169760 non-null  object
 3   ACRS Report Type               169760 non-null  object
 4   Crash Date/Time                169760 non-null  object
 5   Route Type                     152997 non-null  object
 6   Road Name                      154015 non-null  object
 7   Cross-Street Type              152964 non-null  object
 8   Cross-Street Name              154001 non-null  object
 9   Off-Road Description           15743 non-null   object
 10  Municipality                   18852 non-null   object
 11  Related Non-Motorist           5397 non-null    object
 12  Collision Type                 169186 non-null  object
 13  Weather                        156569 non-null  object
 14  Surface Condition              149888 non-null  object
 15  Light                          168347 non-null  object
 16  Traffic Control                144598 non-null  object
 17  Driver Substance Abuse         138840 non-null  object
 18  Non-Motorist Substance Abuse   4268 non-null    object
 19  Person ID                      169760 non-null  object
 20  Driver At Fault                169760 non-null  object
 21  Injury Severity                169760 non-null  object
 22  Circumstance                   30771 non-null   object
 23  Driver Distracted By           169760 non-null  object
 24  Drivers License State          159985 non-null  object
 25  Vehicle ID                     169760 non-null  object
 26  Vehicle Damage Extent          169448 non-null  object
 27  Vehicle First Impact Location  169604 non-null  object
 28  Vehicle Second Impact Location 169504 non-null  object
 29  Vehicle Body Type              167146 non-null  object
 30  Vehicle Movement               169381 non-null  object
 31  Vehicle Continuing Dir         167111 non-null  object
 32  Vehicle Going Dir              167111 non-null  object
 33  Speed Limit                    169760 non-null  int64
 34  Driverless Vehicle             169760 non-null  object
 35  Parked Vehicle                 169760 non-null  object
 36  Vehicle Year                   169760 non-null  int64
 37  Vehicle Make                   169736 non-null  object
 38  Vehicle Model                  169694 non-null  object
 39  Equipment Problems             135954 non-null  object
 40  Latitude                       169760 non-null  float64
 41  Longitude                      169760 non-null  float64
 42  Location                       169760 non-null  object
dtypes: float64(2), int64(2), object(39)
memory usage: 55.7+ MB
```

1.3.3 3.3 Data Cleaning

Removing NULL Values and undesirable columns

In this phase, we conducted a thorough examination of null values within each column and meticulously explored unique values for individual columns to gain a comprehensive understanding of the dataset. Subsequently, we made the decision to eliminate entries with null values, considering the gravity of the prediction model's application in serious cases such as vehicle crashes. Despite the removal of these entries, we still retained a substantial dataset with an estimated row count of around 1 Lakh. Given the critical nature of the model's involvement in addressing vehicle crashes, the decision not to impute values at the moment was intentional. This cautious approach was adopted to ensure that the dataset used for training and evaluation only comprised complete entries, without introducing any potential inaccuracies associated with imputed values. The objective was to maintain the integrity of the data, recognizing the significance of the issue at hand and avoiding any inadvertent manipulation that might compromise the reliability of the model's predictions.

[ ]:
```
data.isnull().sum()
```

[ ]:
```
Report Number                         0
Local Case Number                     0
Agency Name                           0
ACRS Report Type                      0
Crash Date/Time                       0
Route Type                        16763
Road Name                         15745
Cross-Street Type                 16796
Cross-Street Name                 15759
Off-Road Description             154017
Municipality                     150908
Related Non-Motorist             164363
Collision Type                      574
Weather                           13191
Surface Condition                 19872
Light                              1413
Traffic Control                   25162
Driver Substance Abuse            30920
Non-Motorist Substance Abuse     165492
Person ID                             0
Driver At Fault                       0
Injury Severity                       0
Circumstance                     138989
Driver Distracted By                  0
Drivers License State              9775
Vehicle ID                            0
Vehicle Damage Extent               312
Vehicle First Impact Location       156
Vehicle Second Impact Location      256
Vehicle Body Type                  2614
Vehicle Movement                    379
Vehicle Continuing Dir             2649
Vehicle Going Dir                  2649
```

```
Speed Limit                              0
Driverless Vehicle                       0
Parked Vehicle                           0
Vehicle Year                             0
Vehicle Make                            24
Vehicle Model                           66
Equipment Problems                   33806
Latitude                                 0
Longitude                                0
Location                                 0
dtype: int64
```

In the below code, we opted to remove the columns named Off-Road Description, Municipality, Related Non-Motorist, and Non-Motorist Substance Abuse due to a substantial number of missing values. These columns exhibited null values exceeding 85% in the dataset. The decision to drop these columns was made instead of eliminating rows, as the latter approach would have resulted in the loss of a considerable portion of our dataset. Moreover, considering the high percentage of missing values, retaining these columns would not have added meaningful insights to our model. Therefore, we decided to exclude them to enhance the overall quality and relevance of our dataset for subsequent modeling purposes.

[ ]:
```
data.drop(['Off-Road Description','Municipality','Related␣
↪Non-Motorist','Non-Motorist Substance Abuse'], axis=1, inplace=True)
```

[ ]:
```
data['Route Type'].unique()
```

[ ]:
```
array([nan, 'Maryland (State)', 'County', 'US (State)', 'Municipality',
       'Interstate (State)', 'Other Public Roadway', 'Ramp', 'Government',
       'Service Road', 'Unknown'], dtype=object)
```

[ ]:
```
data.dropna(subset=['Route Type'], inplace=True)
```

[ ]:
```
data['Cross-Street Type'].unique()
```

[ ]:
```
array(['Unknown', 'Maryland (State)', 'County', 'US (State)',
       'Other Public Roadway', 'Municipality', 'Ramp', 'Government',
       'Interstate (State)', nan, 'Service Road'], dtype=object)
```

[ ]:
```
data.dropna(subset=['Cross-Street Type'], inplace=True)
```

[ ]:
```
data['Collision Type'].unique()
```

[ ]:
```
array(['STRAIGHT MOVEMENT ANGLE', 'HEAD ON LEFT TURN',
       'SAME DIR REAR END', 'SINGLE VEHICLE', 'HEAD ON',
       'OPPOSITE DIRECTION SIDESWIPE', 'SAME DIRECTION RIGHT TURN',
       'OTHER', 'ANGLE MEETS LEFT TURN', 'SAME DIRECTION SIDESWIPE',
       'SAME DIR BOTH LEFT TURN', 'ANGLE MEETS RIGHT TURN',
```

```
       'SAME DIR REND RIGHT TURN', 'SAME DIRECTION LEFT TURN',
       'ANGLE MEETS LEFT HEAD ON', 'UNKNOWN', 'SAME DIR REND LEFT TURN',
       'OPPOSITE DIR BOTH LEFT TURN', nan], dtype=object)
```

[ ]:
```python
data.dropna(subset=['Collision Type'], inplace=True)
```

[ ]:
```python
data['Weather'].unique()
```

[ ]:
```
array(['CLEAR', 'CLOUDY', 'RAINING', nan, 'UNKNOWN', 'FOGGY', 'OTHER',
       'SNOW', 'BLOWING SNOW', 'WINTRY MIX', 'SEVERE WINDS', 'SLEET',
       'BLOWING SAND, SOIL, DIRT'], dtype=object)
```

[ ]:
```python
data.dropna(subset=['Weather'], inplace=True)
```

[ ]:
```python
data['Surface Condition'].unique()
```

[ ]:
```
array(['DRY', nan, 'WET', 'UNKNOWN', 'WATER(STANDING/MOVING)',
       'MUD, DIRT, GRAVEL', 'ICE', 'SLUSH', 'SNOW', 'OTHER', 'OIL',
       'SAND'], dtype=object)
```

[ ]:
```python
data.dropna(subset=['Surface Condition'], inplace=True)
```

[ ]:
```python
data['Light'].unique()
```

[ ]:
```
array(['DAYLIGHT', 'DUSK', 'DARK -- UNKNOWN LIGHTING', 'DARK LIGHTS ON',
       'DAWN', 'DARK NO LIGHTS', 'UNKNOWN', 'OTHER', nan], dtype=object)
```

[ ]:
```python
data.dropna(subset=['Light'], inplace=True)
```

[ ]:
```python
data['Traffic Control'].unique()
```

[ ]:
```
array(['TRAFFIC SIGNAL', 'NO CONTROLS', 'OTHER', nan, 'STOP SIGN',
       'FLASHING TRAFFIC SIGNAL', 'WARNING SIGN', 'UNKNOWN', 'YIELD SIGN',
       'PERSON', 'SCHOOL ZONE SIGN DEVICE', 'RAILWAY CROSSING DEVICE'],
      dtype=object)
```

[ ]:
```python
data.dropna(subset=['Traffic Control'], inplace=True)
```

[ ]:
```python
data['Driver Substance Abuse'].unique()
```

[ ]:
```
array(['NONE DETECTED', 'ALCOHOL PRESENT', 'UNKNOWN', nan,
       'COMBINED SUBSTANCE PRESENT', 'ALCOHOL CONTRIBUTED',
       'ILLEGAL DRUG CONTRIBUTED', 'MEDICATION PRESENT',
       'MEDICATION CONTRIBUTED', 'ILLEGAL DRUG PRESENT',
       'COMBINATION CONTRIBUTED', 'OTHER'], dtype=object)
```

[ ]:
```python
data.dropna(subset=['Driver Substance Abuse'], inplace=True)
```

[ ]:
```python
data['Circumstance'].unique()
```

[ ]:
```
array([nan, 'ANIMAL, N/A', 'ANIMAL, BACKUP DUE TO NON-RECURRING INCIDENT',
       'N/A, WET', 'BACKUP DUE TO REGULAR CONGESTION, N/A',
       'RAIN, SNOW, WET',
       'SLEET, HAIL, FREEZ. RAIN, TRAFFIC CONTROL DEVICE INOPERATIVE, WET',
       'N/A, ROAD UNDER CONSTRUCTION/MAINTENANCE',
       'DEBRIS OR OBSTRUCTION, RAIN, SNOW, WET',
       'N/A, RUTS, HOLES, BUMPS',
       'N/A, TRAFFIC CONTROL DEVICE INOPERATIVE',
       'V WIPERS|W OTHER ENVIRONMENTAL, WET', 'N/A, RAIN, SNOW',
       'DEBRIS OR OBSTRUCTION, N/A', 'N/A, PHYSICAL OBSTRUCTION(S)',
       'BACKUP DUE TO PRIOR CRASH, N/A', 'N/A, SMOG, SMOKE',
       'SLEET, HAIL, FREEZ. RAIN, WET',
       'N/A, VISION OBSTRUCTION (INCL. BLINDED BY SUN)',
       'RAIN, SNOW, SLEET, HAIL, FREEZ. RAIN, WET',
       'BACKUP DUE TO PRIOR CRASH, RAIN, SNOW, WET',
       'RAIN, SNOW, V EXHAUST SYSTEM|R OTHER ROAD, VISION OBSTRUCTION (INCL.
BLINDED BY SUN), WET',
       'BACKUP DUE TO REGULAR CONGESTION, RAIN, SNOW, WET',
       'N/A, SLEET, HAIL, FREEZ. RAIN',
       'VISION OBSTRUCTION (INCL. BLINDED BY SUN), WET',
       'N/A, V WIPERS|W OTHER ENVIRONMENTAL',
       'BACKUP DUE TO REGULAR CONGESTION, N/A, WET',
       'N/A, TOLL BOOTH/PLAZA RELATED, WET',
       'N/A, V EXHAUST SYSTEM|R OTHER ROAD',
       'RAIN, SNOW, TRAFFIC CONTROL DEVICE INOPERATIVE, WET',
       'ICY OR SNOW-COVERED, SLEET, HAIL, FREEZ. RAIN, WET',
       'ICY OR SNOW-COVERED, RAIN, SNOW, WET',
       'ICY OR SNOW-COVERED, SLEET, HAIL, FREEZ. RAIN',
       'ICY OR SNOW-COVERED, N/A', 'ICY OR SNOW-COVERED, RAIN, SNOW',
       'ICY OR SNOW-COVERED, RAIN, SNOW, SEVERE CROSSWINDS, SLEET, HAIL, FREEZ.
RAIN',
       'ICY OR SNOW-COVERED, VISION OBSTRUCTION (INCL. BLINDED BY SUN)',
       'SEVERE CROSSWINDS, TRAFFIC CONTROL DEVICE INOPERATIVE',
       'N/A, ROAD UNDER CONSTRUCTION/MAINTENANCE, RUTS, HOLES, BUMPS',
       'ICY OR SNOW-COVERED, V WIPERS|W OTHER ENVIRONMENTAL',
       'RAIN, SNOW, VISION OBSTRUCTION (INCL. BLINDED BY SUN), WET',
       'BACKUP DUE TO NON-RECURRING INCIDENT, N/A',
       'DEBRIS OR OBSTRUCTION, N/A, RUTS, HOLES, BUMPS, SHOULDERS LOW, SOFT,
HIGH',
       'ANIMAL, RAIN, SNOW, WET',
       'N/A, ROAD UNDER CONSTRUCTION/MAINTENANCE, WET',
       'RAIN, SNOW, SEVERE CROSSWINDS, WET',
       'N/A, RUTS, HOLES, BUMPS, WET',
       'RAIN, SNOW, TRAFFIC CONTROL DEVICE INOPERATIVE',
       'N/A, TRAFFIC CONTROL DEVICE INOPERATIVE, WET',
```

```
        'SHOULDERS LOW, SOFT, HIGH, VISION OBSTRUCTION (INCL. BLINDED BY SUN)',
        'ICY OR SNOW-COVERED, RAIN, SNOW, SLEET, HAIL, FREEZ. RAIN',
        'DEBRIS OR OBSTRUCTION, V WIPERS|W OTHER ENVIRONMENTAL',
        'ANIMAL, WET',
        'BACKUP DUE TO NON-RECURRING INCIDENT, VISION OBSTRUCTION (INCL. BLINDED
BY SUN)',
        'DEBRIS OR OBSTRUCTION, RAIN, SNOW',
        'N/A, WORN, TRAVEL-POLISHED SURFACE',
        'V WIPERS|W OTHER ENVIRONMENTAL, WET, WORN, TRAVEL-POLISHED SURFACE',
        'SMOG, SMOKE, WET',
        'RAIN, SNOW, V EXHAUST SYSTEM|R OTHER ROAD, V WIPERS|W OTHER
ENVIRONMENTAL, WET',
        'RAIN, SNOW, V EXHAUST SYSTEM|R OTHER ROAD, WET',
        'ICY OR SNOW-COVERED, RAIN, SNOW, SLEET, HAIL, FREEZ. RAIN, WET',
        'N/A, SHOULDERS LOW, SOFT, HIGH',
        'DEBRIS OR OBSTRUCTION, VISION OBSTRUCTION (INCL. BLINDED BY SUN)',
        'PHYSICAL OBSTRUCTION(S), VISION OBSTRUCTION (INCL. BLINDED BY SUN)',
        'RAIN, SNOW, ROAD UNDER CONSTRUCTION/MAINTENANCE',
        'N/A, NON-HIGHWAY WORK', 'SLEET, HAIL, FREEZ. RAIN',
        'ICY OR SNOW-COVERED, N/A, WET',
        'V EXHAUST SYSTEM|R OTHER ROAD, V WIPERS|W OTHER ENVIRONMENTAL',
        'ROAD UNDER CONSTRUCTION/MAINTENANCE, VISION OBSTRUCTION (INCL. BLINDED
BY SUN)',
        'RAIN, SNOW, WORN, TRAVEL-POLISHED SURFACE',
        'ICY OR SNOW-COVERED, V WIPERS|W OTHER ENVIRONMENTAL, WET',
        'ANIMAL, V EXHAUST SYSTEM|R OTHER ROAD',
        'BACKUP DUE TO REGULAR CONGESTION, SLEET, HAIL, FREEZ. RAIN, WET',
        'BACKUP DUE TO PRIOR CRASH, N/A, WET',
        'ANIMAL, ICY OR SNOW-COVERED',
        'N/A, ROAD UNDER CONSTRUCTION/MAINTENANCE, V EXHAUST SYSTEM|R OTHER
ROAD',
        'BACKUP DUE TO REGULAR CONGESTION, N/A, V EXHAUST SYSTEM|R OTHER ROAD',
        'V EXHAUST SYSTEM|R OTHER ROAD, VISION OBSTRUCTION (INCL. BLINDED BY
SUN)',
        'BACKUP DUE TO NON-RECURRING INCIDENT, N/A, ROAD UNDER
CONSTRUCTION/MAINTENANCE',
        'N/A, SEVERE CROSSWINDS',
        'N/A, RAIN, SNOW, VISION OBSTRUCTION (INCL. BLINDED BY SUN)',
        'BACKUP DUE TO NON-RECURRING INCIDENT, ROAD UNDER
CONSTRUCTION/MAINTENANCE, VISION OBSTRUCTION (INCL. BLINDED BY SUN)',
        'RAIN, SNOW, SHOULDERS LOW, SOFT, HIGH',
        'N/A, V EXHAUST SYSTEM|R OTHER ROAD, WET',
        'BACKUP DUE TO NON-RECURRING INCIDENT, N/A, WET',
        'ICY OR SNOW-COVERED, SEVERE CROSSWINDS',
        'SLEET, HAIL, FREEZ. RAIN, V EXHAUST SYSTEM|R OTHER ROAD, WET',
        'DEBRIS OR OBSTRUCTION, N/A, PHYSICAL OBSTRUCTION(S), ROAD UNDER
CONSTRUCTION/MAINTENANCE, RUTS, HOLES, BUMPS',
```

```
       'N/A, WET, WORN, TRAVEL-POLISHED SURFACE',
       'DEBRIS OR OBSTRUCTION, RAIN, SNOW, WET, WORN, TRAVEL-POLISHED SURFACE',
       'N/A, PHYSICAL OBSTRUCTION(S), RUTS, HOLES, BUMPS',
       'BACKUP DUE TO REGULAR CONGESTION, VISION OBSTRUCTION (INCL. BLINDED BY
SUN)',
       'RAIN, SNOW, RUTS, HOLES, BUMPS, WET',
       'BACKUP DUE TO PRIOR CRASH, BACKUP DUE TO REGULAR CONGESTION, N/A',
       'DEBRIS OR OBSTRUCTION, N/A, WET', 'BLOWING SAND, SOIL, DIRT, N/A',
       'RAIN, SNOW, SLEET, HAIL, FREEZ. RAIN, V WIPERS|W OTHER ENVIRONMENTAL,
WET',
       'BACKUP DUE TO NON-RECURRING INCIDENT, BACKUP DUE TO REGULAR CONGESTION,
N/A',
       'ICY OR SNOW-COVERED, RAIN, SNOW, SEVERE CROSSWINDS',
       'RAIN, SNOW, V WIPERS|W OTHER ENVIRONMENTAL, WET',
       'RAIN, SNOW, V EXHAUST SYSTEM|R OTHER ROAD',
       'NON-HIGHWAY WORK, RAIN, SNOW, VISION OBSTRUCTION (INCL. BLINDED BY
SUN)',
       'RAIN, SNOW',
       'BACKUP DUE TO REGULAR CONGESTION, ICY OR SNOW-COVERED, N/A',
       'DEBRIS OR OBSTRUCTION, RAIN, SNOW, V WIPERS|W OTHER ENVIRONMENTAL, WET',
       'RAIN, SNOW, RUTS, HOLES, BUMPS',
       'RUTS, HOLES, BUMPS, VISION OBSTRUCTION (INCL. BLINDED BY SUN)',
       'DEBRIS OR OBSTRUCTION, V WIPERS|W OTHER ENVIRONMENTAL, WET',
       'DEBRIS OR OBSTRUCTION, RAIN, SNOW, SEVERE CROSSWINDS, WET',
       'DEBRIS OR OBSTRUCTION, N/A, PHYSICAL OBSTRUCTION(S)',
       'RAIN, SNOW, ROAD UNDER CONSTRUCTION/MAINTENANCE, SEVERE CROSSWINDS',
       'SEVERE CROSSWINDS, WET',
       'N/A, RAIN, SNOW, V WIPERS|W OTHER ENVIRONMENTAL',
       'BACKUP DUE TO REGULAR CONGESTION, ICY OR SNOW-COVERED, V WIPERS|W OTHER
ENVIRONMENTAL',
       'DEBRIS OR OBSTRUCTION, N/A, ROAD UNDER CONSTRUCTION/MAINTENANCE',
       'ICY OR SNOW-COVERED, RAIN, SNOW, V WIPERS|W OTHER ENVIRONMENTAL',
       'ICY OR SNOW-COVERED, SEVERE CROSSWINDS, V WIPERS|W OTHER ENVIRONMENTAL',
       'N/A, PHYSICAL OBSTRUCTION(S), ROAD UNDER CONSTRUCTION/MAINTENANCE',
       'RAIN, SNOW, SMOG, SMOKE, WET',
       'BACKUP DUE TO REGULAR CONGESTION, N/A, ROAD UNDER
CONSTRUCTION/MAINTENANCE',
       'PHYSICAL OBSTRUCTION(S), RAIN, SNOW, WET',
       'RAIN, SNOW, WET, WORN, TRAVEL-POLISHED SURFACE',
       'BLOWING SAND, SOIL, DIRT, V EXHAUST SYSTEM|R OTHER ROAD, V WIPERS|W
OTHER ENVIRONMENTAL',
       'ICY OR SNOW-COVERED, RAIN, SNOW, V EXHAUST SYSTEM|R OTHER ROAD',
       'BLOWING SAND, SOIL, DIRT, ICY OR SNOW-COVERED, RAIN, SNOW, SEVERE
CROSSWINDS',
       'BACKUP DUE TO REGULAR CONGESTION, ICY OR SNOW-COVERED, RAIN, SNOW',
       'BLOWING SAND, SOIL, DIRT, RAIN, SNOW, SEVERE CROSSWINDS, SLEET, HAIL,
FREEZ. RAIN, WET',
```

```
       'ROAD UNDER CONSTRUCTION/MAINTENANCE, V WIPERS|W OTHER ENVIRONMENTAL',
       'BACKUP DUE TO NON-RECURRING INCIDENT, N/A, NON-HIGHWAY WORK',
       'BACKUP DUE TO REGULAR CONGESTION, N/A, ROAD UNDER
CONSTRUCTION/MAINTENANCE, RUTS, HOLES, BUMPS',
       'BACKUP DUE TO NON-RECURRING INCIDENT, N/A, PHYSICAL OBSTRUCTION(S)',
       'BACKUP DUE TO REGULAR CONGESTION, RAIN, SNOW, V EXHAUST SYSTEM|R OTHER
ROAD, WET',
       'RAIN, SNOW, ROAD UNDER CONSTRUCTION/MAINTENANCE, RUTS, HOLES, BUMPS,
WET',
       'SLEET, HAIL, FREEZ. RAIN, VISION OBSTRUCTION (INCL. BLINDED BY SUN),
WET',
       'BLOWING SAND, SOIL, DIRT, ICY OR SNOW-COVERED',
       'N/A, SHOULDERS LOW, SOFT, HIGH, WET',
       'DEBRIS OR OBSTRUCTION, ICY OR SNOW-COVERED, N/A',
       'DEBRIS OR OBSTRUCTION, SLEET, HAIL, FREEZ. RAIN, TRAFFIC CONTROL DEVICE
INOPERATIVE, VISION OBSTRUCTION (INCL. BLINDED BY SUN)',
       'NON-HIGHWAY WORK, SLEET, HAIL, FREEZ. RAIN, WET',
       'DEBRIS OR OBSTRUCTION, SEVERE CROSSWINDS',
       'BACKUP DUE TO REGULAR CONGESTION, N/A, RUTS, HOLES, BUMPS',
       'DEBRIS OR OBSTRUCTION, ICY OR SNOW-COVERED, RAIN, SNOW',
       'BACKUP DUE TO NON-RECURRING INCIDENT, BACKUP DUE TO PRIOR CRASH, N/A',
       'DEBRIS OR OBSTRUCTION, ICY OR SNOW-COVERED, PHYSICAL OBSTRUCTION(S),
RAIN, SNOW',
       'ICY OR SNOW-COVERED, RAIN, SNOW, SMOG, SMOKE',
       'DEBRIS OR OBSTRUCTION, SLEET, HAIL, FREEZ. RAIN, V WIPERS|W OTHER
ENVIRONMENTAL, WET',
       'SEVERE CROSSWINDS, V EXHAUST SYSTEM|R OTHER ROAD',
       'N/A, ROAD UNDER CONSTRUCTION/MAINTENANCE, RUTS, HOLES, BUMPS, WORN,
TRAVEL-POLISHED SURFACE',
       'BLOWING SAND, SOIL, DIRT, RAIN, SNOW, WET',
       'N/A, V WIPERS|W OTHER ENVIRONMENTAL, VISION OBSTRUCTION (INCL. BLINDED
BY SUN)',
       'ANIMAL, PHYSICAL OBSTRUCTION(S)',
       'DEBRIS OR OBSTRUCTION, SEVERE CROSSWINDS, V EXHAUST SYSTEM|R OTHER
ROAD',
       'PHYSICAL OBSTRUCTION(S), RAIN, SNOW',
       'ANIMAL, SLEET, HAIL, FREEZ. RAIN, WET'], dtype=object)
```

[ ]:
```
data.drop(['Circumstance'], axis=1, inplace=True)
```

[ ]:
```
data['Drivers License State'].unique()
```

[ ]:
```
array(['MD', 'CA', nan, 'DC', 'VA', 'NY', 'XX', 'TX', 'NJ', 'GA', 'TN',
       'WA', 'ND', 'MO', 'PA', 'MS', 'NC', 'NM', 'FL', 'UT', 'IN', 'WV',
       'AZ', 'AL', 'MI', 'CT', 'NH', 'IL', 'DE', 'OH', 'NE', 'WI', 'MN',
       'US', 'CO', 'HI', 'NF', 'NV', 'ME', 'LA', 'AB', 'MH', 'AR', 'OK',
       'MB', 'SC', 'ON', 'MA', 'KY', 'OR', 'PR', 'IA', 'ID', 'IT', 'MT',
```

```
       'AK', 'RI', 'YT', 'QC', 'VI', 'NL', 'WY', 'KS', 'VT', 'UM', 'NS',
       'BC', 'GU', 'AS'], dtype=object)
```

[ ]:
```
data['Drivers License State'].fillna('unknown', inplace=True)
```

[ ]:
```
data['Drivers License State'].unique()
```

[ ]:
```
array(['MD', 'CA', 'unknown', 'DC', 'VA', 'NY', 'XX', 'TX', 'NJ', 'GA',
       'TN', 'WA', 'ND', 'MO', 'PA', 'MS', 'NC', 'NM', 'FL', 'UT', 'IN',
       'WV', 'AZ', 'AL', 'MI', 'CT', 'NH', 'IL', 'DE', 'OH', 'NE', 'WI',
       'MN', 'US', 'CO', 'HI', 'NF', 'NV', 'ME', 'LA', 'AB', 'MH', 'AR',
       'OK', 'MB', 'SC', 'ON', 'MA', 'KY', 'OR', 'PR', 'IA', 'ID', 'IT',
       'MT', 'AK', 'RI', 'YT', 'QC', 'VI', 'NL', 'WY', 'KS', 'VT', 'UM',
       'NS', 'BC', 'GU', 'AS'], dtype=object)
```

[ ]:
```
data.dropna(subset=['Vehicle Damage Extent'], inplace=True)
```

[ ]:
```
data['Vehicle First Impact Location'].unique()
```

[ ]:
```
array(['THREE OCLOCK', 'TWELVE OCLOCK', 'SIX OCLOCK', 'FOUR OCLOCK',
       'UNKNOWN', 'NINE OCLOCK', 'ONE OCLOCK', 'TEN OCLOCK',
       'SEVEN OCLOCK', 'ELEVEN OCLOCK', 'EIGHT OCLOCK', 'TWO OCLOCK',
       'FIVE OCLOCK', 'NON-COLLISION', 'ROOF TOP', 'UNDERSIDE', nan],
      dtype=object)
```

[ ]:
```
data.dropna(subset=['Vehicle First Impact Location'], inplace=True)
```

[ ]:
```
data['Vehicle Second Impact Location'].unique()
```

[ ]:
```
array(['TWO OCLOCK', 'TWELVE OCLOCK', 'SIX OCLOCK', 'FOUR OCLOCK',
       'FIVE OCLOCK', 'UNKNOWN', 'NINE OCLOCK', 'ONE OCLOCK',
       'TEN OCLOCK', 'SEVEN OCLOCK', 'ELEVEN OCLOCK', 'EIGHT OCLOCK',
       'THREE OCLOCK', 'NON-COLLISION', 'UNDERSIDE', 'ROOF TOP', nan],
      dtype=object)
```

[ ]:
```
data.dropna(subset=['Vehicle Second Impact Location'], inplace=True)
```

[ ]:
```
data['Equipment Problems'].unique()
```

[ ]:
```
array(['NO MISUSE', 'UNKNOWN', nan, 'OTHER', 'AIR BAG FAILED',
       'STRAP/TETHER LOOSE', 'NOT STREPPED RIGHT', 'BELTS/ANCHORS BROKE',
       'BELT(S) MISUSED', 'FACING WRONG WAY', 'SIZE/TYPE IMPROPER'],
      dtype=object)
```

[ ]:
```
data.dropna(subset=['Equipment Problems'], inplace=True)
```

[ ]:
```
data['Vehicle Body Type'].unique()
```

```
[ ]: array(['PASSENGER CAR', 'PICKUP TRUCK', '(SPORT) UTILITY VEHICLE',
            'TRANSIT BUS', 'VAN', 'MOTORCYCLE', 'UNKNOWN', 'TRUCK TRACTOR',
            nan, 'POLICE VEHICLE/NON EMERGENCY',
            'MEDIUM/HEAVY TRUCKS 3 AXLES (OVER 10,000LBS (4,536KG))',
            'OTHER LIGHT TRUCKS (10,000LBS (4,536KG) OR LESS)',
            'CARGO VAN/LIGHT TRUCK 2 AXLES (OVER 10,000LBS (4,536 KG))',
            'POLICE VEHICLE/EMERGENCY', 'OTHER BUS', 'MOPED', 'SCHOOL BUS',
            'RECREATIONAL VEHICLE', 'OTHER', 'AMBULANCE/EMERGENCY',
            'AUTOCYCLE', 'STATION WAGON', 'SNOWMOBILE',
            'FIRE VEHICLE/EMERGENCY', 'ALL TERRAIN VEHICLE (ATV)',
            'FIRE VEHICLE/NON EMERGENCY', 'AMBULANCE/NON EMERGENCY',
            'FARM VEHICLE', 'LOW SPEED VEHICLE', 'CROSS COUNTRY BUS',
            'LIMOUSINE'], dtype=object)
```

```
[ ]: data.dropna(subset=['Vehicle Body Type'], inplace=True)
```

```
[ ]: data['Vehicle Movement'].unique()
```

```
[ ]: array(['MAKING LEFT TURN', 'ACCELERATING', 'STARTING FROM LANE',
            'STOPPED IN TRAFFIC LANE', 'SLOWING OR STOPPING',
            'MOVING CONSTANT SPEED', 'MAKING RIGHT TURN', 'UNKNOWN',
            'MAKING U TURN', 'CHANGING LANES', 'PASSING', 'PARKING',
            'LEAVING TRAFFIC LANE', 'BACKING', 'NEGOTIATING A CURVE',
            'ENTERING TRAFFIC LANE', 'STARTING FROM PARKED',
            'RIGHT TURN ON RED', 'SKIDDING', nan, 'OTHER', 'PARKED',
            'DRIVERLESS MOVING VEH.'], dtype=object)
```

```
[ ]: data.dropna(subset=['Vehicle Movement'], inplace=True)
```

```
[ ]: data['Vehicle Continuing Dir'].unique()
```

```
[ ]: array(['East', 'North', 'West', 'South', 'Unknown', nan], dtype=object)
```

```
[ ]: data.dropna(subset=['Vehicle Continuing Dir'], inplace=True)
```

```
[ ]: data['Vehicle Make'].unique()
```

```
[ ]: array(['GMC', 'FORD', 'KIA', …, 'ICRB', 'INTR', 'GENE'], dtype=object)
```

```
[ ]: data.dropna(subset=['Vehicle Make'], inplace=True)
```

```
[ ]: data['Vehicle Model'].unique()
```

```
[ ]: array(['TK', 'F150', 'SW', …, 'DURANGOQ', 'CORR', 'RENAGADE'],
           dtype=object)
```

```
[ ]: data.dropna(subset=['Vehicle Model'], inplace=True)
```

```
[ ]: data['Driver Distracted By'].unique()
```

```
[ ]: array(['NOT DISTRACTED', 'LOOKED BUT DID NOT SEE', 'UNKNOWN',
            'INATTENTIVE OR LOST IN THOUGHT', 'OTHER DISTRACTION',
            'USING OTHER DEVICE CONTROLS INTEGRAL TO VEHICLE',
            'TEXTING FROM A CELLULAR PHONE',
            'DISTRACTED BY OUTSIDE PERSON OBJECT OR EVENT',
            'OTHER CELLULAR PHONE RELATED', 'NO DRIVER PRESENT',
            'TALKING OR LISTENING TO CELLULAR PHONE', 'BY OTHER OCCUPANTS',
            'ADJUSTING AUDIO AND OR CLIMATE CONTROLS', 'EATING OR DRINKING',
            'OTHER ELECTRONIC DEVICE (NAVIGATIONAL PALM PILOT)',
            'BY MOVING OBJECT IN VEHICLE',
            'USING DEVICE OBJECT BROUGHT INTO VEHICLE', 'SMOKING RELATED',
            'DIALING CELLULAR PHONE'], dtype=object)
```

Listing all the columns that are currently available in the dataset.

```
[ ]: data.columns
```

```
[ ]: Index(['Report Number', 'Local Case Number', 'Agency Name', 'ACRS Report Type',
            'Crash Date/Time', 'Route Type', 'Road Name', 'Cross-Street Type',
            'Cross-Street Name', 'Collision Type', 'Weather', 'Surface Condition',
            'Light', 'Traffic Control', 'Driver Substance Abuse', 'Person ID',
            'Driver At Fault', 'Injury Severity', 'Driver Distracted By',
            'Drivers License State', 'Vehicle ID', 'Vehicle Damage Extent',
            'Vehicle First Impact Location', 'Vehicle Second Impact Location',
            'Vehicle Body Type', 'Vehicle Movement', 'Vehicle Continuing Dir',
            'Vehicle Going Dir', 'Speed Limit', 'Driverless Vehicle',
            'Parked Vehicle', 'Vehicle Year', 'Vehicle Make', 'Vehicle Model',
            'Equipment Problems', 'Latitude', 'Longitude', 'Location'],
           dtype='object')
```

We have loaded the selected set of columns into a new dataset called “data1” in order to avoid any kind of discrepancies in the future.

```
[ ]: selected_columns = [
         'Crash Date/Time', 'Route Type', 'Cross-Street Type', 'Collision Type',
         'Weather', 'Surface Condition', 'Light', 'Traffic Control',
         'Driver Substance Abuse', 'Driver At Fault', 'Driver Distracted By',
         'Drivers License State', 'Vehicle First Impact Location',
         'Vehicle Second Impact Location', 'Vehicle Body Type', 'Vehicle Movement',
         'Vehicle Continuing Dir', 'Vehicle Going Dir', 'Speed Limit',
         'Equipment Problems', 'Injury Severity', 'Vehicle Damage Extent',␣
      ↪'Longitude', 'Latitude'
     ]
     data1 = data[selected_columns]
```

When we were examining the values in the ‘Crash Date/Time’ column, we recognized that the values in this column encompass the entire datetime stamp, resulting in the difficult usage of the values in the column, in technical terms. Consequently, we made the decision to decode this datetime stamp into distinct components such as year and month, aiming to gain a more detailed comprehension of the datetime values. Subsequently, as the 'Crash Date/Time' column had been deconstructed into various components, we opted to drop it from the dataset. Additionally, we chose to exclude the 'Minutes' and 'Seconds' columns, deeming their inclusion as features less pertinent for the predictive modeling process. This decision was taken in order to streamline and optimize the dataset for more effective model training and interpretation.

[ ]:
```
data1['Crash Date/Time']=pd.to_datetime(data1['Crash Date/Time'])
```

```
/var/tmp/ipykernel_6156/2797296375.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1['Crash Date/Time']=pd.to_datetime(data1['Crash Date/Time'])
```

[ ]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 86483 entries, 1 to 169758
Data columns (total 24 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Crash Date/Time                 86483 non-null  datetime64[ns]
 1   Route Type                      86483 non-null  object
 2   Cross-Street Type               86483 non-null  object
 3   Collision Type                  86483 non-null  object
 4   Weather                         86483 non-null  object
 5   Surface Condition               86483 non-null  object
 6   Light                           86483 non-null  object
 7   Traffic Control                 86483 non-null  object
 8   Driver Substance Abuse          86483 non-null  object
 9   Driver At Fault                 86483 non-null  object
 10  Driver Distracted By            86483 non-null  object
 11  Drivers License State           86483 non-null  object
 12  Vehicle First Impact Location   86483 non-null  object
 13  Vehicle Second Impact Location  86483 non-null  object
 14  Vehicle Body Type               86483 non-null  object
 15  Vehicle Movement                86483 non-null  object
 16  Vehicle Continuing Dir          86483 non-null  object
 17  Vehicle Going Dir               86483 non-null  object
 18  Speed Limit                     86483 non-null  int64
 19  Equipment Problems              86483 non-null  object
 20  Injury Severity                 86483 non-null  object
 21  Vehicle Damage Extent           86483 non-null  object
```

```
 22  Longitude                        86483 non-null  float64
 23  Latitude                         86483 non-null  float64
dtypes: datetime64[ns](1), float64(2), int64(1), object(20)
memory usage: 16.5+ MB
```

[ ]:
```
data1.loc[:, 'Crash Date/Time'] = pd.to_datetime(data1['Crash Date/Time'],␣
↪format='%m/%d/%Y %I:%M:%S %p')

data1.loc[:, 'Year'] = data1['Crash Date/Time'].dt.year
data1.loc[:, 'Month'] = data1['Crash Date/Time'].dt.month
data1.loc[:, 'Day'] = data1['Crash Date/Time'].dt.day
data1.loc[:, 'Hour'] = data1['Crash Date/Time'].dt.hour
data1.loc[:, 'Minute'] = data1['Crash Date/Time'].dt.minute
data1.loc[:, 'Second'] = data1['Crash Date/Time'].dt.second
data1.loc[:, 'AM/PM'] = data1['Crash Date/Time'].dt.strftime('%p')
```

```
/var/tmp/ipykernel_6156/2139270624.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1.loc[:, 'Year'] = data1['Crash Date/Time'].dt.year
/var/tmp/ipykernel_6156/2139270624.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1.loc[:, 'Month'] = data1['Crash Date/Time'].dt.month
/var/tmp/ipykernel_6156/2139270624.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1.loc[:, 'Day'] = data1['Crash Date/Time'].dt.day
/var/tmp/ipykernel_6156/2139270624.py:6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1.loc[:, 'Hour'] = data1['Crash Date/Time'].dt.hour
/var/tmp/ipykernel_6156/2139270624.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

```
See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1.loc[:, 'Minute'] = data1['Crash Date/Time'].dt.minute
/var/tmp/ipykernel_6156/2139270624.py:8: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1.loc[:, 'Second'] = data1['Crash Date/Time'].dt.second
/var/tmp/ipykernel_6156/2139270624.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  data1.loc[:, 'AM/PM'] = data1['Crash Date/Time'].dt.strftime('%p')
```

[ ]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 86483 entries, 1 to 169758
Data columns (total 31 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Crash Date/Time                 86483 non-null  datetime64[ns]
 1   Route Type                      86483 non-null  object
 2   Cross-Street Type               86483 non-null  object
 3   Collision Type                  86483 non-null  object
 4   Weather                         86483 non-null  object
 5   Surface Condition               86483 non-null  object
 6   Light                           86483 non-null  object
 7   Traffic Control                 86483 non-null  object
 8   Driver Substance Abuse          86483 non-null  object
 9   Driver At Fault                 86483 non-null  object
 10  Driver Distracted By            86483 non-null  object
 11  Drivers License State           86483 non-null  object
 12  Vehicle First Impact Location   86483 non-null  object
 13  Vehicle Second Impact Location  86483 non-null  object
 14  Vehicle Body Type               86483 non-null  object
 15  Vehicle Movement                86483 non-null  object
 16  Vehicle Continuing Dir          86483 non-null  object
 17  Vehicle Going Dir               86483 non-null  object
 18  Speed Limit                     86483 non-null  int64
 19  Equipment Problems              86483 non-null  object
 20  Injury Severity                 86483 non-null  object
 21  Vehicle Damage Extent           86483 non-null  object
 22  Longitude                       86483 non-null  float64
```

```
 23  Latitude                        86483 non-null  float64
 24  Year                            86483 non-null  int32
 25  Month                           86483 non-null  int32
 26  Day                             86483 non-null  int32
 27  Hour                            86483 non-null  int32
 28  Minute                          86483 non-null  int32
 29  Second                          86483 non-null  int32
 30  AM/PM                           86483 non-null  object
dtypes: datetime64[ns](1), float64(2), int32(6), int64(1), object(21)
memory usage: 19.1+ MB
```

[ ]:
```
data1 = data1.drop(columns=['Crash Date/Time','Minute','Second'])
```

Let's take a look at the top few rows from the newly constructed dataset.

[ ]:
```
data1.head(5)
```

We will now review the variable types in the new dataset, data1 and check if there are any more null values present in it.

[ ]:
```
data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 86483 entries, 1 to 169758
Data columns (total 24 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   Crash Date/Time                 86483 non-null  object
 1   Route Type                      86483 non-null  object
 2   Cross-Street Type               86483 non-null  object
 3   Collision Type                  86483 non-null  object
 4   Weather                         86483 non-null  object
 5   Surface Condition               86483 non-null  object
 6   Light                           86483 non-null  object
 7   Traffic Control                 86483 non-null  object
 8   Driver Substance Abuse          86483 non-null  object
 9   Driver At Fault                 86483 non-null  object
 10  Driver Distracted By            86483 non-null  object
 11  Drivers License State           86483 non-null  object
 12  Vehicle First Impact Location   86483 non-null  object
 13  Vehicle Second Impact Location  86483 non-null  object
 14  Vehicle Body Type               86483 non-null  object
 15  Vehicle Movement                86483 non-null  object
 16  Vehicle Continuing Dir          86483 non-null  object
 17  Vehicle Going Dir               86483 non-null  object
 18  Speed Limit                     86483 non-null  int64
 19  Equipment Problems              86483 non-null  object
 20  Injury Severity                 86483 non-null  object
```

```
 21  Vehicle Damage Extent           86483 non-null  object
 22  Longitude                       86483 non-null  float64
 23  Latitude                        86483 non-null  float64
dtypes: float64(2), int64(1), object(21)
memory usage: 16.5+ MB
```

[ ]:
```
data1.isnull().sum()
```

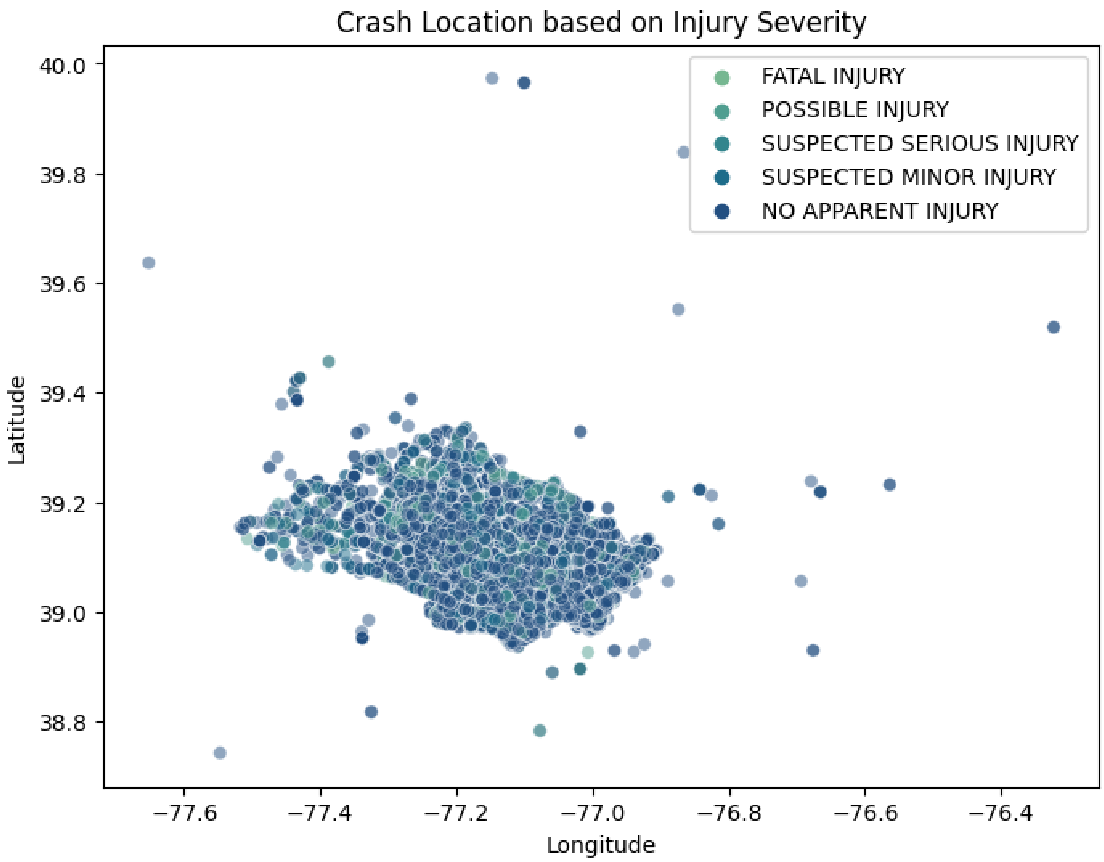
[ ]:
```
Crash Date/Time                  0
Route Type                       0
Cross-Street Type                0
Collision Type                   0
Weather                          0
Surface Condition                0
Light                            0
Traffic Control                  0
Driver Substance Abuse           0
Driver At Fault                  0
Driver Distracted By             0
Drivers License State            0
Vehicle First Impact Location    0
Vehicle Second Impact Location   0
Vehicle Body Type                0
Vehicle Movement                 0
Vehicle Continuing Dir           0
Vehicle Going Dir                0
Speed Limit                      0
Equipment Problems               0
Injury Severity                  0
Vehicle Damage Extent            0
Longitude                        0
Latitude                         0
dtype: int64
```

1.3.4 3.4 Data Visualization

Presented here are visualizations that have provided insights into the interrelationships among various columns in the dataset. These visual representations have been instrumental in enhancing our understanding of how different attributes within the dataset correlate and interact.

[ ]:
```
#Create a scatter plot to visualize the geographic distribution of crashes␣
↪based on injury severity.
plt.figure(figsize=(8, 6))
hue_order = ['FATAL INJURY', 'POSSIBLE INJURY', 'SUSPECTED SERIOUS INJURY',␣
↪'SUSPECTED MINOR INJURY', 'NO APPARENT INJURY']
sns.scatterplot(data=data1, x='Longitude', y='Latitude', hue='Injury Severity',␣
↪hue_order=hue_order,
```

```
                sizes=(1, 100), alpha=0.5, palette='crest')
plt.title('Crash Location based on Injury Severity')
plt.legend(scatterpoints=1)
plt.show()
```



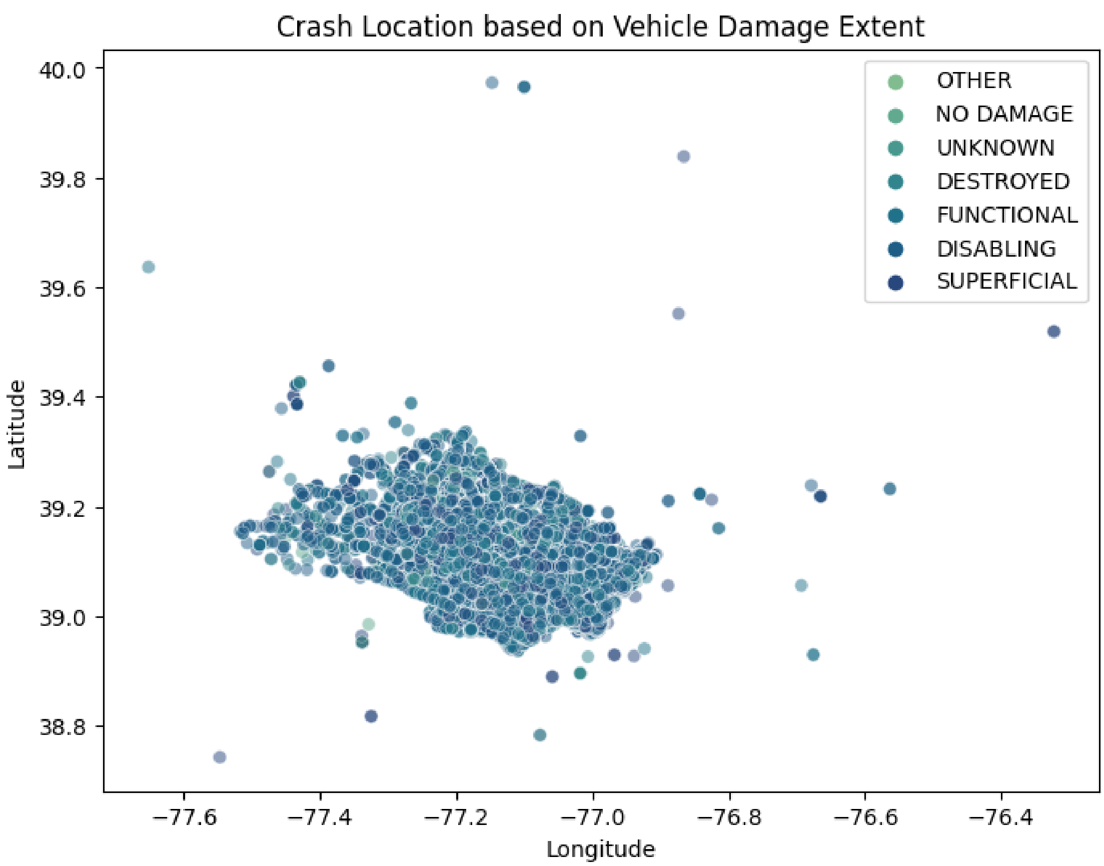
The scatterplot above provides a geographical representation of crash locations, offering valuable insights into the correlation between injury severity and the geographic coordinates (longitude and latitude) of each incident.

The hue variation in the scatterplot reflects different levels of injury severity, allowing for a visual categorization of crash incidents. Brighter hues signify higher injury severity, while darker hues indicate less severe outcomes. This color-coded approach enhances the interpretability of the spatial distribution of crashes and the associated injury severity.

Since the dataset contains data of only maryland, the scatter plots for are concentrated in one area.

By combining geographic information with injury severity data, this visualization serves as a powerful tool for stakeholders involved in traffic safety analysis and policymaking. It facilitates the identification of hotspots or areas prone to more severe accidents, aiding in the development of targeted strategies for accident prevention and response. The scatterplot is a visually intuitive way to comprehend the complex interplay between location, injury severity, and crash density in the context of road safety.

```
[ ]: # Create a scatter plot to visualize the geographic distribution of crashes␣
     ↪based on the extent of vehicle damage
     plt.figure(figsize=(8, 6))
     hue_order=['OTHER', 'NO DAMAGE', 'UNKNOWN', 'DESTROYED', 'FUNCTIONAL',␣
     ↪'DISABLING', 'SUPERFICIAL']
     sns.scatterplot(data=data1, x='Longitude', y='Latitude', hue='Vehicle Damage␣
     ↪Extent', hue_order=hue_order,
                     sizes=(1, 100), alpha=0.5, palette='crest')
     plt.title('Crash Location based on Vehicle Damage Extent')
     plt.legend(scatterpoints=1)
     plt.show()
```

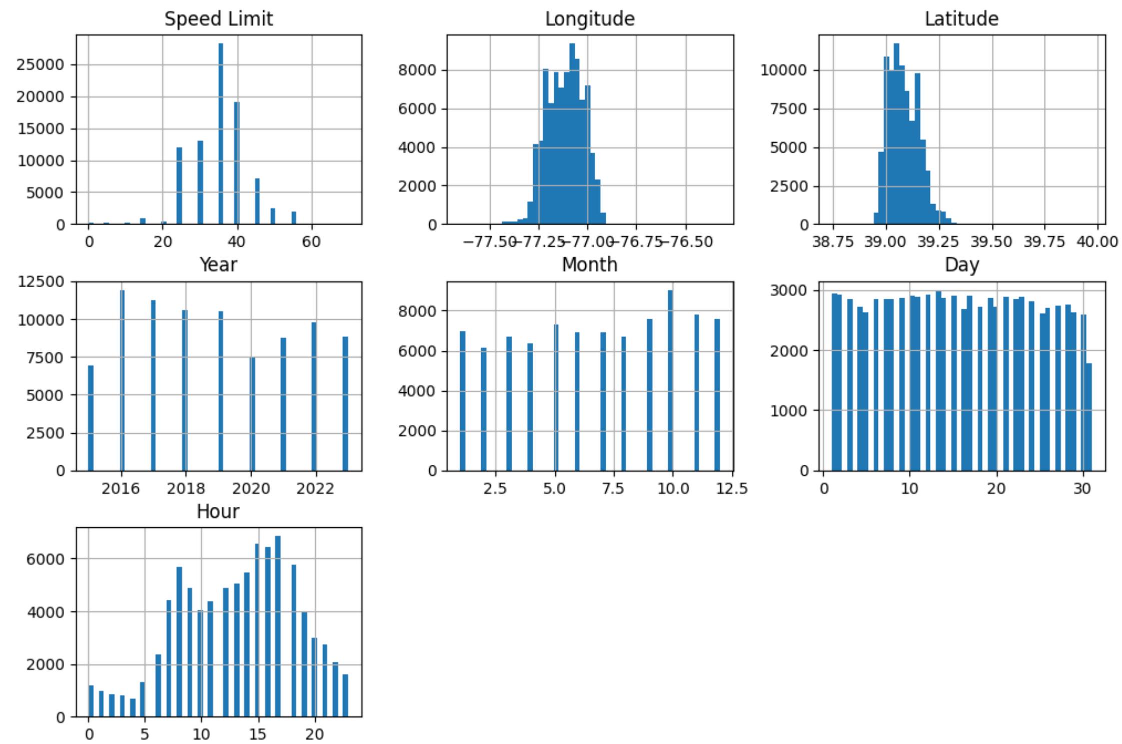


The scatterplot above provides a geographical representation of crash locations, offering insights into the correlation between the extent of vehicle damage and the geographic coordinates (longitude and latitude) of each incident. The color-coded approach enhances the interpretability of the spatial distribution of crashes and the associated vehicle damage.

Since the dataset contains data of only maryland, the scatter plots for are concentrated in one area.

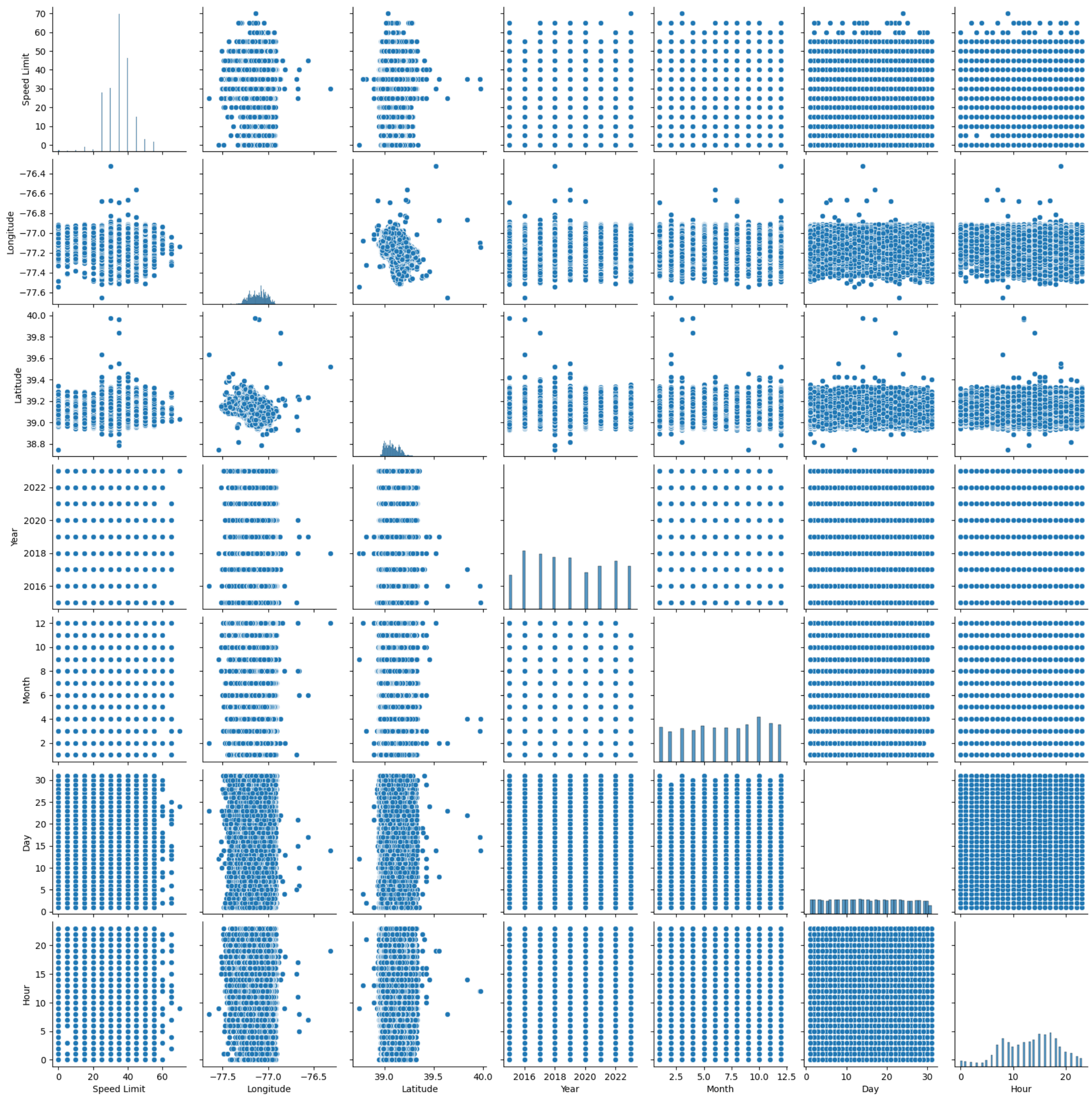
By combining geographic information with vehicle damage extent data, this visualization serves as a powerful tool for stakeholders involved in traffic safety analysis and policymaking. It facilitates the identification of hotspots or areas prone to more severe vehicle damage, aiding in the development of targeted strategies for accident prevention and response. The scatterplot is a visually intuitive way to comprehend the complex interplay between location, vehicle damage extent, and crash density in the context of road safety.

[ ]:
```
# Generate histograms for each numerical column in the 'data1' DataFrame with␣
↪50 bins and a figure size of 12 by 8.
data1.hist(bins=50, figsize=(12, 8))
plt.show()
```



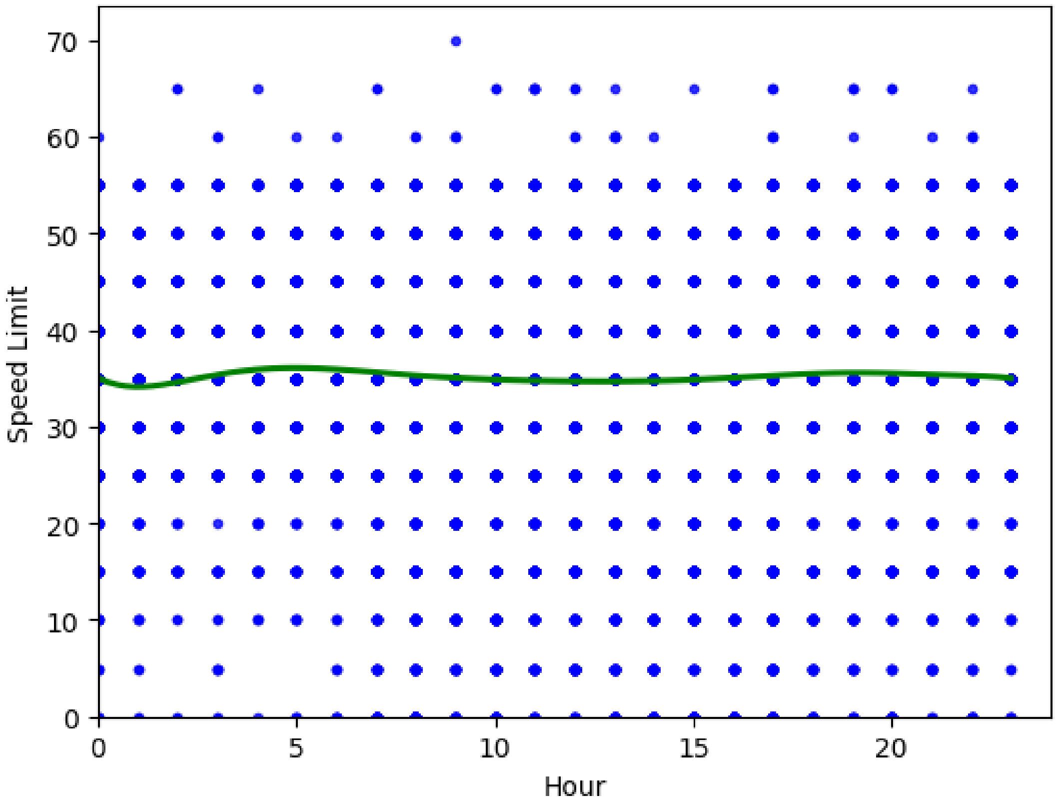
The histograms above offer a comprehensive perspective on the distribution patterns of numerical variables within the dataset. Each histogram visually represents the dispersion of values across various ranges. Notably, some graphs exhibit a normal distribution, while others showcase a more dispersed pattern. Collectively, these histograms provide a clear overview of the count distribution for each numerical variable, offering valuable insights into the dataset's characteristics and highlighting variations in the distribution of values across different features.

[ ]:
```
sns.pairplot(data1);
```



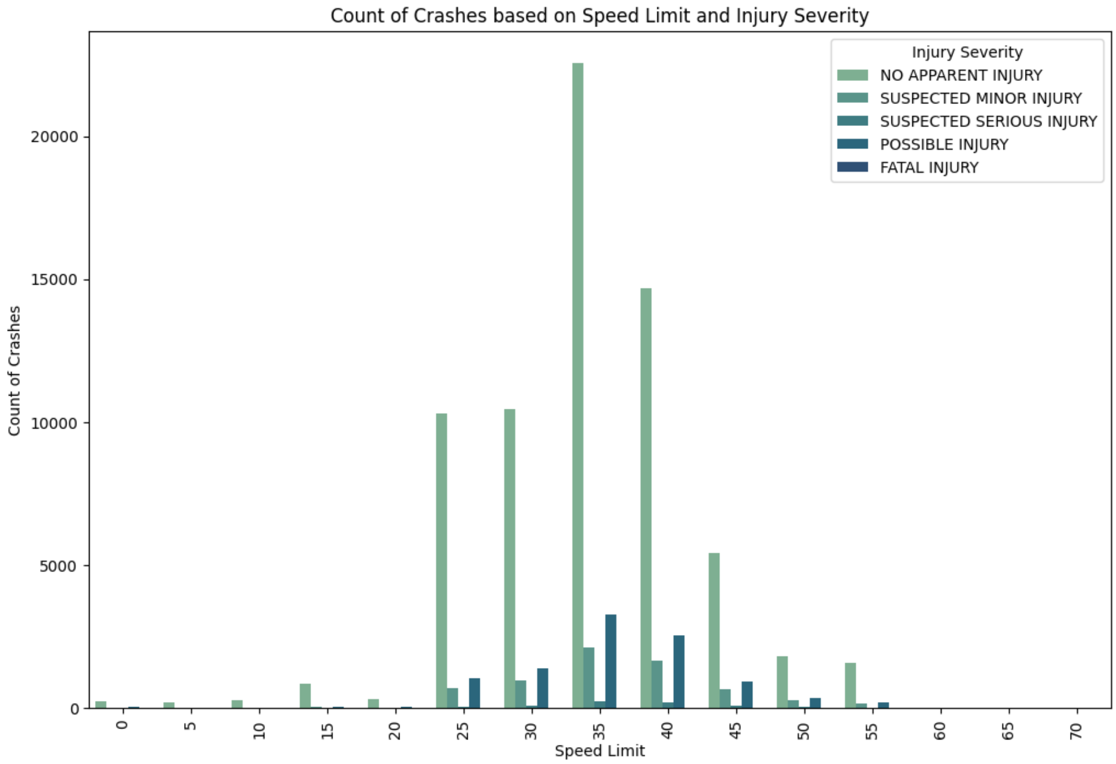
The pairplot illustrates the relationships among numerical variables, aiding in the identification of increased associations between them. This is valuable for discerning the influence one variable may have on another, offering insights crucial for enhancing our machine learning model.

```
[ ]: # Scatter plot with a polynomial regression line depicting the relationship
     ↪between 'Speed Limit' and 'Hour'.
     sns.regplot(x=data1['Hour'], y=data1['Speed Limit'], order=10, color='green',
     ↪ci=None, scatter_kws={'color':'blue', 's':9})
     plt.xlim(0,24)
     plt.ylim(ymin=0);
```



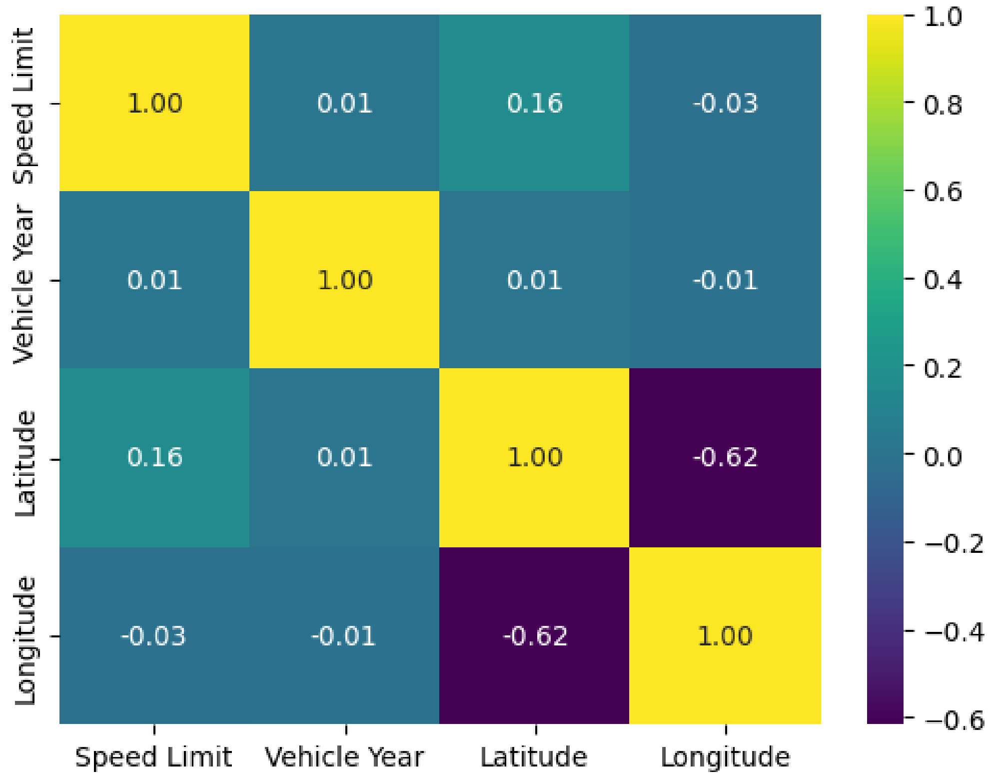
The presented plot illustrates the relationship between speed limits and the hour of the day. Notably, the speed limit appears relatively consistent with minor fluctuations, particularly around noon. From this analysis, it can be inferred that accidents attributed to speed limits exhibit limited correlation with the hour of the day when solely considering these two variables. However, it's essential to recognize that this relationship may evolve when additional variables are introduced into the analysis.

```
[ ]: # Create a countplot to visualize the distribution of crashes based on 'Speed␣
     ↪Limit' and 'Injury Severity'.
     plt.figure(figsize=(12, 8))
     sns.countplot(data=data1, x='Speed Limit', hue='Injury Severity',␣
     ↪palette='crest')
     plt.ylabel('Count of Crashes')
     plt.title('Count of Crashes based on Speed Limit and Injury Severity')
     plt.xticks(rotation=90);
```



The depicted graphs showcase the overall count of crashes corresponding to each speed limit. Notably, the highest count is observed at a speed limit of approximately 33. This insightful representation offers a comprehensive view of the distribution of crashes across different speed limits, highlighting the specific speed limit where the occurrence of crashes is most prevalent.

[ ]:
```
# Generate a heatmap to visualize the correlation matrix of numerical variables␣
 ↪in the dataset.
corr_matrix = data.corr(numeric_only=True)
sns.heatmap(corr_matrix, annot=True, fmt='.2f', cmap='viridis');
```



This heatmap effectively visualizes the correlation among numeric variables within the dataset. The color spectrum and numerical values depicted in each cell offer a clear indication of the degree of relevance between different variables and how they mutually influence one another.

1.4 4. Machine Learning Techniques

1.4.1 4.1 Split the Data

In the below code snippet, we are preparing our dataset for training and testing by separating features (X) from the target variables for injury severity (y1) and vehicle damage extent (y2).

[ ]:
```
# Splitting the dataset into features (X) and target variables for injury␣
↪severity (y1) and vehicle damage extent (y2).
X = data1.drop(["Injury Severity", "Vehicle Damage Extent"], axis=1)
y1 = data1["Injury Severity"].copy()
y2 = data1["Vehicle Damage Extent"].copy()
X_train, X_test, y1_train, y1_test, y2_train, y2_test = train_test_split(
    X, y1, y2, test_size=0.25, random_state=3)
```

We are separating the features (independent variables) and target variables ("Injury Severity" and "Vehicle Damage Extent") from the data1 DataFrame. The train_test_split function from the scikit-learn library is then used to randomly split the data into training and testing sets for both target variables, allocating 25% of the data for testing and maintaining reproducibility with a specified random seed of 3. The resulting variables (X_train, X_test, y1_train, y1_test, y2_train, y2_test) are further utilized for training and evaluating machine learning models.

1.4.2 4.2 Create a Pipeline

In the below code, a preprocessing pipeline is established to handle numerical and categorical features independently. For numerical features, it involves imputing missing values using the median and scaling features using StandardScaler. For categorical features, missing values are imputed with the most frequent values, and one-hot encoding is applied. The overall preprocessing is encapsulated in a ColumnTransformer, allowing for a streamlined transformation of the dataset. The preprocessing steps are applied separately to training and testing sets (X_train and X_test). The use of an imputer, even after removing null values in your dataset, is a practice aimed at ensuring the robustness and generalization ability of your machine learning model. In a real-world scenario, the machine learning model may be deployed to make predictions on new, incoming data in real-time. The imputer ensures that your model can handle missing values in real-time predictions, maintaining its accuracy and reliability.

[ ]:
```
# Creating a preprocessing pipeline to handle numerical and categorical␣
↪features separately.
set_config(display='diagram') # Shows the pipeline graphically when printed

#Calling the categorical and numerical variables to cat_attribs and num_attribs␣
↪respectively
cat_attribs = ['Route Type', 'Cross-Street Type', 'Collision Type', 'Weather',␣
↪'Surface Condition', 'Light', 'Traffic Control',
               'Driver Substance Abuse', 'Driver At Fault', 'Driver Distracted␣
↪By',
               'Drivers License State', 'Vehicle First Impact Location',
               'Vehicle Second Impact Location', 'Vehicle Body Type',
               'Vehicle Movement', 'Vehicle Continuing Dir', 'Vehicle Going␣
↪Dir',
               'Speed Limit', 'Equipment Problems', 'AM/PM', 'Year', 'Month',
               'Day', 'Hour']

num_attribs = ['Longitude', 'Latitude']

# Numerical pipeline
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Categorical pipeline
cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('cat_encoder', OneHotEncoder(sparse_output=False))
])
```

```python
# Full preprocessing pipeline
prep_pipeline = ColumnTransformer([
    ('num', num_pipeline, num_attribs),
    ('cat', cat_pipeline, cat_attribs)
], verbose_feature_names_out=False)
prep_pipeline.set_output(transform='pandas')

# We apply the preprocessing as a separate step and work with the transformed␣
 ↪data
X_train_prepd = prep_pipeline.fit_transform(X_train)

# preprocess the X_test as well
X_test_prepd = prep_pipeline.transform(X_test)
prep_pipeline
```

```
[ ]: ColumnTransformer(transformers=[('num',
                                 Pipeline(steps=[('imputer',
SimpleImputer(strategy='median')),
                                                 ('scaler', StandardScaler())]),
                                 ['Longitude', 'Latitude']),
                                ('cat',
                                 Pipeline(steps=[('imputer',
SimpleImputer(strategy='most_frequent')),
                                                 ('cat_encoder',
OneHotEncoder(sparse_output=False))]),
                                 ['Route Type', 'Cross-Street Type',
                                  'Collision Type', 'Weather',
                                  'Surface…n', 'Light',
                                  'Traffic Control', 'Driver Substance Abuse',
                                  'Driver At Fault', 'Driver Distracted By',
                                  'Drivers License State',
                                  'Vehicle First Impact Location',
                                  'Vehicle Second Impact Location',
                                  'Vehicle Body Type', 'Vehicle Movement',
                                  'Vehicle Continuing Dir', 'Vehicle Going Dir',
                                  'Speed Limit', 'Equipment Problems', 'AM/PM',
                                  'Year', 'Month', 'Day', 'Hour'])],
                  verbose_feature_names_out=False)
```

We are setting up a preprocessing pipeline for our dataset, we are handling both numerical and categorical features in the pipeline. The set_config(display='diagram') line configures the display option to show the pipeline graphically when printed. The features are divided into two groups: num_attribs for numerical features and cat_attribs for categorical features. We are constructing two separate pipelines for numerical and categorical data, applying imputation for missing values and scaling for numerical features, and imputation and one-hot encoding for categorical features. These pipelines are combined using a ColumnTransformer to create an overall preprocessing pipeline (prep_pipeline). This pipeline is then applied to transform both the training (X_train) and testing (X_test) datasets. The result is a transformed dataset that can be further used for training and evaluating our predictive model, with numerical features imputed and scaled, and categorical features imputed and one-hot encoded. The graphical representation of the pipeline is presented in order to visualize the preprocessing steps.

The below code snippet aims to compare unique values in categorical columns between the training and test sets. It iterates through each categorical attribute, identifying and printing any new categories introduced in the test set that were not present in the training set. This step was done for ensuring consistency and compatibility between the two sets during model evaluation as few splits did not take each category in each split.

[ ]:
```
# Examine unique values in categorical columns for training set
unique_values_train = {}
for col in cat_attribs:
    unique_values_train[col] = set(X_train[col].unique())

# Examine unique values in categorical columns for test set
unique_values_test = {}
for col in cat_attribs:
    unique_values_test[col] = set(X_test[col].unique())

# Print the unique values for each categorical column
for col in cat_attribs:
    new_categories_in_test = unique_values_test[col] - unique_values_train[col]
    if new_categories_in_test:
        print(f"New categories in '{col}' for test set:␣
↪{new_categories_in_test}")
```

With the above code, we are examining and comparing unique categorical values between the training set (`X_train`) and the test set (`X_test`). The process iterates through the specified categorical columns (`cat_attribs`) and creates dictionaries (`unique_values_train` and `unique_values_test`) that store the unique values for each column in the training and test sets, respectively. We are then printing any new categories found in the test set that are not present in the training set for each categorical column. This process helps to identify and alert users to any unforeseen categories in the test set, ensuring consistency in categorical values between training and testing data, which is crucial for accurate our prediction model.

[ ]:
```
# Displaying the shapes of training and testing sets for features and target␣
↪variables.
X_train.shape, X_test.shape, y1_train.shape, y1_test.shape,y2_train.
↪shape,y2_test.shape
```

[ ]:
```
((64862, 26), (21621, 26), (64862,), (21621,), (64862,), (21621,))
```

Checking and printing the shapes of the training, testing features and training and testing of the two prediction tasks present in our model.

```
[ ]: # Comparing the number of columns obtained from 'get_feature_names_out()' with␣
     ↪the actual number of columns in the transformed training set.
     print("Number of columns from get_feature_names_out():", len(prep_pipeline.
      ↪get_feature_names_out()))
     print("Number of columns in X_train_transformed:", X_train_prepd.shape[1])
```

```
Number of columns from get_feature_names_out(): 378
Number of columns in X_train_transformed: 378
```

We are printing the number of columns in the transformed training data after applying a preprocessing pipeline. The first line uses the get_feature_names_out() method from the pipeline (prep_pipeline) to obtain the names of the transformed features and then prints the count of the feature names. The second line prints the number of columns directly from the shape of the transformed training data (X_train_prepd). Comparing these two counts helps ensure consistency and correctness in the preprocessing steps, confirming that the expected number of features is obtained after applying the pipeline to the training data.

We can view the tranformed columns below

```
[ ]: # Displaying the column names obtained from 'get_feature_names_out()' method in␣
     ↪the preprocessing pipeline.
     print("Column names from get_feature_names_out():", prep_pipeline.
      ↪get_feature_names_out())
```

```
Column names from get_feature_names_out(): ['Longitude' 'Latitude' 'Route
Type_County' 'Route Type_Government'
 'Route Type_Interstate (State)' 'Route Type_Maryland (State)'
 'Route Type_Municipality' 'Route Type_Other Public Roadway'
 'Route Type_Ramp' 'Route Type_Service Road' 'Route Type_US (State)'
 'Route Type_Unknown' 'Cross-Street Type_County'
 'Cross-Street Type_Government' 'Cross-Street Type_Interstate (State)'
 'Cross-Street Type_Maryland (State)' 'Cross-Street Type_Municipality'
 'Cross-Street Type_Other Public Roadway' 'Cross-Street Type_Ramp'
 'Cross-Street Type_Service Road' 'Cross-Street Type_US (State)'
 'Cross-Street Type_Unknown' 'Collision Type_ANGLE MEETS LEFT HEAD ON'
 'Collision Type_ANGLE MEETS LEFT TURN'
 'Collision Type_ANGLE MEETS RIGHT TURN' 'Collision Type_HEAD ON'
 'Collision Type_HEAD ON LEFT TURN'
 'Collision Type_OPPOSITE DIR BOTH LEFT TURN'
 'Collision Type_OPPOSITE DIRECTION SIDESWIPE' 'Collision Type_OTHER'
 'Collision Type_SAME DIR BOTH LEFT TURN'
 'Collision Type_SAME DIR REAR END'
 'Collision Type_SAME DIR REND LEFT TURN'
 'Collision Type_SAME DIR REND RIGHT TURN'
 'Collision Type_SAME DIRECTION LEFT TURN'
 'Collision Type_SAME DIRECTION RIGHT TURN'
 'Collision Type_SAME DIRECTION SIDESWIPE' 'Collision Type_SINGLE VEHICLE'
 'Collision Type_STRAIGHT MOVEMENT ANGLE' 'Collision Type_UNKNOWN'
 'Weather_BLOWING SAND, SOIL, DIRT' 'Weather_BLOWING SNOW' 'Weather_CLEAR'
```

```
'Weather_CLOUDY' 'Weather_FOGGY' 'Weather_OTHER' 'Weather_RAINING'
'Weather_SEVERE WINDS' 'Weather_SLEET' 'Weather_SNOW' 'Weather_UNKNOWN'
'Weather_WINTRY MIX' 'Surface Condition_DRY' 'Surface Condition_ICE'
'Surface Condition_MUD, DIRT, GRAVEL' 'Surface Condition_OIL'
'Surface Condition_OTHER' 'Surface Condition_SAND'
'Surface Condition_SLUSH' 'Surface Condition_SNOW'
'Surface Condition_UNKNOWN' 'Surface Condition_WATER(STANDING/MOVING)'
'Surface Condition_WET' 'Light_DARK -- UNKNOWN LIGHTING'
'Light_DARK LIGHTS ON' 'Light_DARK NO LIGHTS' 'Light_DAWN'
'Light_DAYLIGHT' 'Light_DUSK' 'Light_OTHER' 'Light_UNKNOWN'
'Traffic Control_FLASHING TRAFFIC SIGNAL' 'Traffic Control_NO CONTROLS'
'Traffic Control_OTHER' 'Traffic Control_PERSON'
'Traffic Control_RAILWAY CROSSING DEVICE'
'Traffic Control_SCHOOL ZONE SIGN DEVICE' 'Traffic Control_STOP SIGN'
'Traffic Control_TRAFFIC SIGNAL' 'Traffic Control_UNKNOWN'
'Traffic Control_WARNING SIGN' 'Traffic Control_YIELD SIGN'
'Driver Substance Abuse_ALCOHOL CONTRIBUTED'
'Driver Substance Abuse_ALCOHOL PRESENT'
'Driver Substance Abuse_COMBINATION CONTRIBUTED'
'Driver Substance Abuse_COMBINED SUBSTANCE PRESENT'
'Driver Substance Abuse_ILLEGAL DRUG CONTRIBUTED'
'Driver Substance Abuse_ILLEGAL DRUG PRESENT'
'Driver Substance Abuse_MEDICATION CONTRIBUTED'
'Driver Substance Abuse_MEDICATION PRESENT'
'Driver Substance Abuse_NONE DETECTED' 'Driver Substance Abuse_OTHER'
'Driver Substance Abuse_UNKNOWN' 'Driver At Fault_No'
'Driver At Fault_Unknown' 'Driver At Fault_Yes'
'Driver Distracted By_ADJUSTING AUDIO AND OR CLIMATE CONTROLS'
'Driver Distracted By_BY MOVING OBJECT IN VEHICLE'
'Driver Distracted By_BY OTHER OCCUPANTS'
'Driver Distracted By_DIALING CELLULAR PHONE'
'Driver Distracted By_DISTRACTED BY OUTSIDE PERSON OBJECT OR EVENT'
'Driver Distracted By_EATING OR DRINKING'
'Driver Distracted By_INATTENTIVE OR LOST IN THOUGHT'
'Driver Distracted By_LOOKED BUT DID NOT SEE'
'Driver Distracted By_NO DRIVER PRESENT'
'Driver Distracted By_NOT DISTRACTED'
'Driver Distracted By_OTHER CELLULAR PHONE RELATED'
'Driver Distracted By_OTHER DISTRACTION'
'Driver Distracted By_OTHER ELECTRONIC DEVICE (NAVIGATIONAL PALM PILOT)'
'Driver Distracted By_SMOKING RELATED'
'Driver Distracted By_TALKING OR LISTENING TO CELLULAR PHONE'
'Driver Distracted By_TEXTING FROM A CELLULAR PHONE'
'Driver Distracted By_UNKNOWN'
'Driver Distracted By_USING DEVICE OBJECT BROUGHT INTO VEHICLE'
'Driver Distracted By_USING OTHER DEVICE CONTROLS INTEGRAL TO VEHICLE'
'Drivers License State_AB' 'Drivers License State_AK'
'Drivers License State_AL' 'Drivers License State_AR'
```

```
'Drivers License State_AS' 'Drivers License State_AZ'
'Drivers License State_BC' 'Drivers License State_CA'
'Drivers License State_CO' 'Drivers License State_CT'
'Drivers License State_DC' 'Drivers License State_DE'
'Drivers License State_FL' 'Drivers License State_GA'
'Drivers License State_GU' 'Drivers License State_HI'
'Drivers License State_IA' 'Drivers License State_ID'
'Drivers License State_IL' 'Drivers License State_IN'
'Drivers License State_IT' 'Drivers License State_KS'
'Drivers License State_KY' 'Drivers License State_LA'
'Drivers License State_MA' 'Drivers License State_MB'
'Drivers License State_MD' 'Drivers License State_ME'
'Drivers License State_MH' 'Drivers License State_MI'
'Drivers License State_MN' 'Drivers License State_MO'
'Drivers License State_MS' 'Drivers License State_NC'
'Drivers License State_ND' 'Drivers License State_NE'
'Drivers License State_NF' 'Drivers License State_NH'
'Drivers License State_NJ' 'Drivers License State_NL'
'Drivers License State_NM' 'Drivers License State_NV'
'Drivers License State_NY' 'Drivers License State_OH'
'Drivers License State_OK' 'Drivers License State_ON'
'Drivers License State_OR' 'Drivers License State_PA'
'Drivers License State_PR' 'Drivers License State_QC'
'Drivers License State_RI' 'Drivers License State_SC'
'Drivers License State_TN' 'Drivers License State_TX'
'Drivers License State_UM' 'Drivers License State_US'
'Drivers License State_UT' 'Drivers License State_VA'
'Drivers License State_VI' 'Drivers License State_VT'
'Drivers License State_WA' 'Drivers License State_WI'
'Drivers License State_WV' 'Drivers License State_WY'
'Drivers License State_XX' 'Drivers License State_YT'
'Drivers License State_unknown'
'Vehicle First Impact Location_EIGHT OCLOCK'
'Vehicle First Impact Location_ELEVEN OCLOCK'
'Vehicle First Impact Location_FIVE OCLOCK'
'Vehicle First Impact Location_FOUR OCLOCK'
'Vehicle First Impact Location_NINE OCLOCK'
'Vehicle First Impact Location_NON-COLLISION'
'Vehicle First Impact Location_ONE OCLOCK'
'Vehicle First Impact Location_ROOF TOP'
'Vehicle First Impact Location_SEVEN OCLOCK'
'Vehicle First Impact Location_SIX OCLOCK'
'Vehicle First Impact Location_TEN OCLOCK'
'Vehicle First Impact Location_THREE OCLOCK'
'Vehicle First Impact Location_TWELVE OCLOCK'
'Vehicle First Impact Location_TWO OCLOCK'
'Vehicle First Impact Location_UNDERSIDE'
'Vehicle First Impact Location_UNKNOWN'
```

'Vehicle Second Impact Location_EIGHT OCLOCK'
'Vehicle Second Impact Location_ELEVEN OCLOCK'
'Vehicle Second Impact Location_FIVE OCLOCK'
'Vehicle Second Impact Location_FOUR OCLOCK'
'Vehicle Second Impact Location_NINE OCLOCK'
'Vehicle Second Impact Location_NON-COLLISION'
'Vehicle Second Impact Location_ONE OCLOCK'
'Vehicle Second Impact Location_ROOF TOP'
'Vehicle Second Impact Location_SEVEN OCLOCK'
'Vehicle Second Impact Location_SIX OCLOCK'
'Vehicle Second Impact Location_TEN OCLOCK'
'Vehicle Second Impact Location_THREE OCLOCK'
'Vehicle Second Impact Location_TWELVE OCLOCK'
'Vehicle Second Impact Location_TWO OCLOCK'
'Vehicle Second Impact Location_UNDERSIDE'
'Vehicle Second Impact Location_UNKNOWN'
'Vehicle Body Type_(SPORT) UTILITY VEHICLE'
'Vehicle Body Type_ALL TERRAIN VEHICLE (ATV)'
'Vehicle Body Type_AMBULANCE/EMERGENCY'
'Vehicle Body Type_AMBULANCE/NON EMERGENCY' 'Vehicle Body Type_AUTOCYCLE'
'Vehicle Body Type_CARGO VAN/LIGHT TRUCK 2 AXLES (OVER 10,000LBS (4,536 KG))'
'Vehicle Body Type_CROSS COUNTRY BUS' 'Vehicle Body Type_FARM VEHICLE'
'Vehicle Body Type_FIRE VEHICLE/EMERGENCY'
'Vehicle Body Type_FIRE VEHICLE/NON EMERGENCY'
'Vehicle Body Type_LIMOUSINE' 'Vehicle Body Type_LOW SPEED VEHICLE'
'Vehicle Body Type_MEDIUM/HEAVY TRUCKS 3 AXLES (OVER 10,000LBS (4,536KG))'
'Vehicle Body Type_MOPED' 'Vehicle Body Type_MOTORCYCLE'
'Vehicle Body Type_OTHER' 'Vehicle Body Type_OTHER BUS'
'Vehicle Body Type_OTHER LIGHT TRUCKS (10,000LBS (4,536KG) OR LESS)'
'Vehicle Body Type_PASSENGER CAR' 'Vehicle Body Type_PICKUP TRUCK'
'Vehicle Body Type_POLICE VEHICLE/EMERGENCY'
'Vehicle Body Type_POLICE VEHICLE/NON EMERGENCY'
'Vehicle Body Type_RECREATIONAL VEHICLE' 'Vehicle Body Type_SCHOOL BUS'
'Vehicle Body Type_SNOWMOBILE' 'Vehicle Body Type_STATION WAGON'
'Vehicle Body Type_TRANSIT BUS' 'Vehicle Body Type_TRUCK TRACTOR'
'Vehicle Body Type_UNKNOWN' 'Vehicle Body Type_VAN'
'Vehicle Movement_ACCELERATING' 'Vehicle Movement_BACKING'
'Vehicle Movement_CHANGING LANES'
'Vehicle Movement_DRIVERLESS MOVING VEH.'
'Vehicle Movement_ENTERING TRAFFIC LANE'
'Vehicle Movement_LEAVING TRAFFIC LANE'
'Vehicle Movement_MAKING LEFT TURN' 'Vehicle Movement_MAKING RIGHT TURN'
'Vehicle Movement_MAKING U TURN' 'Vehicle Movement_MOVING CONSTANT SPEED'
'Vehicle Movement_NEGOTIATING A CURVE' 'Vehicle Movement_OTHER'
'Vehicle Movement_PARKING' 'Vehicle Movement_PASSING'
'Vehicle Movement_RIGHT TURN ON RED' 'Vehicle Movement_SKIDDING'
'Vehicle Movement_SLOWING OR STOPPING'
'Vehicle Movement_STARTING FROM LANE'

```
'Vehicle Movement_STARTING FROM PARKED'
'Vehicle Movement_STOPPED IN TRAFFIC LANE' 'Vehicle Movement_UNKNOWN'
'Vehicle Continuing Dir_East' 'Vehicle Continuing Dir_North'
'Vehicle Continuing Dir_South' 'Vehicle Continuing Dir_Unknown'
'Vehicle Continuing Dir_West' 'Vehicle Going Dir_East'
'Vehicle Going Dir_North' 'Vehicle Going Dir_South'
'Vehicle Going Dir_Unknown' 'Vehicle Going Dir_West' 'Speed Limit_0'
'Speed Limit_5' 'Speed Limit_10' 'Speed Limit_15' 'Speed Limit_20'
'Speed Limit_25' 'Speed Limit_30' 'Speed Limit_35' 'Speed Limit_40'
'Speed Limit_45' 'Speed Limit_50' 'Speed Limit_55' 'Speed Limit_60'
'Speed Limit_65' 'Speed Limit_70' 'Equipment Problems_AIR BAG FAILED'
'Equipment Problems_BELT(S) MISUSED'
'Equipment Problems_BELTS/ANCHORS BROKE'
'Equipment Problems_FACING WRONG WAY' 'Equipment Problems_NO MISUSE'
'Equipment Problems_NOT STREPPED RIGHT' 'Equipment Problems_OTHER'
'Equipment Problems_SIZE/TYPE IMPROPER'
'Equipment Problems_STRAP/TETHER LOOSE' 'Equipment Problems_UNKNOWN'
'AM/PM_AM' 'AM/PM_PM' 'Year_2015' 'Year_2016' 'Year_2017' 'Year_2018'
'Year_2019' 'Year_2020' 'Year_2021' 'Year_2022' 'Year_2023' 'Month_1'
'Month_2' 'Month_3' 'Month_4' 'Month_5' 'Month_6' 'Month_7' 'Month_8'
'Month_9' 'Month_10' 'Month_11' 'Month_12' 'Day_1' 'Day_2' 'Day_3'
'Day_4' 'Day_5' 'Day_6' 'Day_7' 'Day_8' 'Day_9' 'Day_10' 'Day_11'
'Day_12' 'Day_13' 'Day_14' 'Day_15' 'Day_16' 'Day_17' 'Day_18' 'Day_19'
'Day_20' 'Day_21' 'Day_22' 'Day_23' 'Day_24' 'Day_25' 'Day_26' 'Day_27'
'Day_28' 'Day_29' 'Day_30' 'Day_31' 'Hour_0' 'Hour_1' 'Hour_2' 'Hour_3'
'Hour_4' 'Hour_5' 'Hour_6' 'Hour_7' 'Hour_8' 'Hour_9' 'Hour_10' 'Hour_11'
'Hour_12' 'Hour_13' 'Hour_14' 'Hour_15' 'Hour_16' 'Hour_17' 'Hour_18'
'Hour_19' 'Hour_20' 'Hour_21' 'Hour_22' 'Hour_23']
```

We are using the get_feature_names_out() method from the pipeline (prep_pipeline) to return the list of column names corresponding to the transformed features. The code helps us to visualize the feature importances of our model. Feature importances represent the contribution of each feature in making predictions. We are sorting the features based on their importance, and then selecting a subset of it to display in a bar chart using Matplotlib. The x-axis represents the feature importance scores, and the y-axis displays the names of the corresponding features. This visualization provides a clear understanding of which features have the most significant impact on the model's predictions, aiding in feature selection, interpretation, and model evaluation.

The next step that we followed was to analyse and decide whether we wanted to build one model for both the output variables or two different ones for each variable

[ ]:
```python
# Create a contingency table
contingency_table = pd.crosstab(data1["Injury Severity"], data1["Vehicle Damage Extent"])

# Perform the chi-square test
chi2, p, _, _ = chi2_contingency(contingency_table)
```

```
print(f"Chi-square statistic: {chi2}")
print(f"P-value: {p}")
```

```
Chi-square statistic: 9080.499465747742
P-value: 0.0
```

In this code, a contingency table is created using the pd.crosstab function to examine the relationship between 'Injury Severity' and 'Vehicle Damage Extent'. The chi-square statistic of 9080.50 and a p-value of 0.0 indicate a highly significant association between the variables 'Injury Severity' and 'Vehicle Damage Extent.' In statistical terms, a low p-value (approaching zero) suggests strong evidence against the null hypothesis, which posits that the variables are independent. Therefore, we reject the null hypothesis and conclude that there is a significant relationship between the severity of injuries and the extent of vehicle damage in the dataset

[ ]:
```
# Calculate Cramér's V
num_obs = np.sum(contingency_table)
cramers_v = np.sqrt(chi2 / (num_obs * (min(contingency_table.shape) - 1)))
print(f"Cramér's V: {cramers_v}")
```

```
Cramér's V: Vehicle Damage Extent
DESTROYED      0.726677
DISABLING      0.253077
FUNCTIONAL     0.312938
NO DAMAGE      0.954445
OTHER          7.629438
SUPERFICIAL    0.345051
UNKNOWN        1.075935
dtype: float64
```

Cramér's V is a measure of association between categorical variables, and the values provided for each category of 'Vehicle Damage Extent' suggest the strength of association with 'Injury Severity. These statistical measures help determine the strength and significance of the relationship between the two variables.We can observe that there is high association between few classes while there is less for others even the when seen as whole the two variables showed a high association. Due to this ambiguity, we chose to handle the two output varibales seperately.

1.4.3 4.3 Machine Learning Models

4.3.1 For y1 Prediction - Injury Severity Initially, our approach involved experimenting with various models on the dataset, with the intention of selecting the most suitable one based on performance metrics. However, several models yielded similar results, prompting us to delve into hyperparameter tuning and feature selection. The goal was to refine and enhance the models, enabling us to make a more informed decision on the optimal choice. Given the gravity of the situation underlying the dataset, it became imperative to construct the best possible model. Below, we outline each model's tuning process and, ultimately, the criteria used to select the superior model among them.

Logistic Regression Model

```
[ ]: model_y1 = LogisticRegression(multi_class='multinomial', solver='lbfgs',␣
      ↪max_iter=100)
     model_y1.fit(X_train_prepd, y1_train)

     y1_pred = model_y1.predict(X_train_prepd)

     balanced_accuracy_y1 = balanced_accuracy_score(y1_train, y1_pred)

     # Precision is computed using the average parameter
     precision_y1 = precision_score(y1_train, y1_pred, average='weighted')

     # Cross-validation scores
     cv_score_y1 = cross_val_score(model_y1, X_train_prepd, y1_train, cv=5,␣
      ↪scoring='accuracy')

     print(f"Accuracy (Injury Severity): {accuracy_score(y1_train, y1_pred)}")
     print(f"Balanced Accuracy (Injury Severity): {balanced_accuracy_y1}")
     print(f"Precision (Injury Severity): {precision_y1}")
     print(f"Cross-Validation Accuracy (Injury Severity): {cv_score_y1.mean()}")
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
```

```
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Accuracy (Injury Severity): 0.8007461996238168
Balanced Accuracy (Injury Severity): 0.2789701548246751
Precision (Injury Severity): 0.707686552156298
Cross-Validation Accuracy (Injury Severity): 0.8001757605676383
```

We are training a Logistic Regression model on the preprocessed training data (X_train_prepd) to predict the target variable "Injury Severity" (y1_train). The trained model's predictions on the training data are stored in y1_pred. It creates a Logistic Regression model (model_y1) with specified parameters for multiclass classification (multi_class='multinomial'), solver (solver='lbfgs'), and maximum number of iterations (max_iter=100). Performance metrics such as accuracy, balanced accuracy, precision, and cross-validation accuracy are then computed and printed. The balance accuracy function calculates the balanced accuracy score for the two arguments, the true labels (y1_train) and the predicted labels (y1_pred). The precision score takes three arguments: the true labels (y1_train), the predicted labels (y1_pred), and the average parameter. We are using "weighted" for average parameter which helps in calculating the average precision with respect to the number of instances in each class. This will result higher weight to classes with fewer instances, making it useful for an imbalanced dataset. The cross_val_score function performs cross-validation, evaluating the model's performance on different subsets of the training data. We are using a 5-fold cross-validation (cv=5) and calculating the accuracy (scoring='accuracy').

Analysis:

Accuracy (Injury Severity): 0.8007 The accuracy score of 0.8007 indicates that approximately 80.07% of the predictions made by the logistic regression model on the training data are correct. It gives an overall measure of the model's correctness. Balanced Accuracy (Injury Severity): 0.2790

The balanced accuracy score of 0.2790 This takes into account the imbalances in the distribution of classes. A low balanced accuracy suggests that the model might struggle with classes that are underrepresented in the dataset. Precision (Injury Severity): 0.7077

The precision score of 0.7077 This reflects the model's ability to correctly identify instances of each class, taking into account the weighted average. This indicates that, on average, 70.77% of the instances predicted as positive are indeed positive.

Cross-Validation Accuracy (Injury Severity): 0.8002 The cross-validation accuracy score of 0.8002 provides an estimate of the model's generalization performance. It suggests that the model performs consistently well across different subsets of the training data.

Over All Analysis:

The over all fit of the model to the data is good but it might struggle on imbalanced distribution of classes. Further, we can notice that there were many errors thrown during the process which might show that the model is not able to properly fit the data,

Feature Importance for the logistic regression model

The below code snippet prints the coefficients and intercepts of the logistic regression model trained to predict 'Injury Severity.' The coefficients represent the weight assigned to each feature for each class, while the intercept provides the model's baseline. Understanding these values is essential for interpreting the contribution of each feature to the prediction of different classes of injury severity.

[ ]:
```
# Displaying coefficients and intercepts of the logistic regression model for␣
↪'Injury Severity.'
print("Coefficients:")
for i, class_coef in enumerate(model_y1.coef_):
    print(f"Class {i} Coefficients:")
    for j, coef in enumerate(class_coef):
        print(f"    Feature {j}: {coef}")

# Print intercept
print("Intercept:")
for i, intercept in enumerate(model_y1.intercept_):
    print(f"Class {i} Intercept: {intercept}")
```

```
Coefficients:
Class 0 Coefficients:
    Feature 0: -0.04830223593954226
```

```
Feature 1: 0.056387440381744106
Feature 2: -0.1358363816489309
Feature 3: -0.019795747258649448
Feature 4: -0.07928466145362692
Feature 5: 0.13838223025513732
Feature 6: -0.14769145167065661
Feature 7: -0.037208535074596275
Feature 8: -0.021784117316238843
Feature 9: -0.0009354618717800556
Feature 10: -0.18320002853521905
Feature 11: -0.00021369957835091512
Feature 12: -0.09806253488493788
Feature 13: 0.04943661162405069
Feature 14: -0.016398319402408077
Feature 15: -0.30047291572833706
Feature 16: -0.12240072750527196
Feature 17: -0.12284574246946073
Feature 18: 0.10370102166336648
Feature 19: -0.0038755584295150597
Feature 20: -0.03871715769363085
Feature 21: 0.06206746867323235
Feature 22: 0.07865485464056442
Feature 23: -0.06662914767690283
Feature 24: -0.028154049231550498
Feature 25: 0.16511696022205072
Feature 26: 0.4815589724926684
Feature 27: -0.008158193537646066
Feature 28: -0.06618985246947456
Feature 29: -0.1826771548302729
Feature 30: -0.018234405301946373
Feature 31: -0.7210285783610566
Feature 32: -0.015534924987550576
Feature 33: -0.015888806826838794
Feature 34: 0.1056380781082051
Feature 35: -0.0772698665164358
Feature 36: -0.26076579420715035
Feature 37: 0.13236013683312178
Feature 38: 0.036268426042307594
Feature 39: -0.026634508545001996
Feature 40: -8.302817213788589e-05
Feature 41: -0.00445457263005463
Feature 42: 0.250246829334822
Feature 43: -0.22354287384800084
Feature 44: -0.025432757803389862
Feature 45: -0.006653374822133361
Feature 46: -0.375006747332157
Feature 47: -0.00604450478422671
Feature 48: -0.004829576957663341
```

```
Feature 49: -0.033348898772254484
Feature 50: -0.04751569530491054
Feature 51: -0.010902653060806433
Feature 52: 0.12345528656945828
Feature 53: -0.030866175767612296
Feature 54: -0.0013243361520873683
Feature 55: -0.0006808062753619192
Feature 56: -0.002993445824443056
Feature 57: -0.00012812539496702158
Feature 58: -0.006812658838694696
Feature 59: -0.02369447308223625
Feature 60: -0.047565871865620404
Feature 61: -0.0015640209573661611
Feature 62: -0.4953932265639775
Feature 63: -0.015837455669512286
Feature 64: -0.14223375871460459
Feature 65: 0.15789005265241507
Feature 66: 0.07339752454936502
Feature 67: -0.6616489210140717
Feature 68: 0.1522711537145363
Feature 69: -0.0074927758012922115
Feature 70: -0.043913673869743895
Feature 71: 0.11741732300476129
Feature 72: -0.2033820554762062
Feature 73: -0.05776908701109179
Feature 74: -0.005932626823994692
Feature 75: -0.0009698346347811463
Feature 76: -0.0004787017753811122
Feature 77: -0.09895196788137185
Feature 78: -0.2558366206226534
Feature 79: -0.0308386890206733
Feature 80: 0.09178965945660572
Feature 81: -0.04261525336812504
Feature 82: -0.08082492450778196
Feature 83: -0.10818815692279138
Feature 84: 0.18513895143931736
Feature 85: 0.18098160188872078
Feature 86: -0.011215467105653855
Feature 87: 0.0638935210779129
Feature 88: -0.005469567059048303
Feature 89: 0.0902746829441235
Feature 90: -1.3158730059822463
Feature 91: -0.003213654472455752
Feature 92: 0.5169281645469948
Feature 93: -0.1088124042864449
Feature 94: 0.6288639672813151
Feature 95: -1.0076194171477726
Feature 96: -0.0025531627177554535
```

Feature 97: -0.004092051999021618
Feature 98: -0.009861378255231992
Feature 99: -0.0010739896821586096
Feature 100: -0.021091246058085763
Feature 101: -0.006136452854197098
Feature 102: -0.10493926211391948
Feature 103: -0.26617396951650146
Feature 104: -0.009215256120653013
Feature 105: -0.9424826090658553
Feature 106: -0.010700652477155802
Feature 107: -0.10391238095192318
Feature 108: -0.00864664591791571
Feature 109: -0.0020248948393515103
Feature 110: -0.006901235456848878
Feature 111: -0.00279944588875563
Feature 112: 1.0190381533009916
Feature 113: -0.001641025008100863
Feature 114: -0.002360348530477113
Feature 115: -0.00023561002542441485
Feature 116: -0.0011140940666203173
Feature 117: -0.002219360277923378
Feature 118: -0.0007728967542098009
Feature 119: -5.78009343883781e-05
Feature 120: -0.0023543311300481
Feature 121: -3.3034459008684525e-05
Feature 122: -0.011457204699606106
Feature 123: -0.0048213925260685055
Feature 124: -0.002793591908520895
Feature 125: -0.03397597217113536
Feature 126: -0.0038517832793392185
Feature 127: -0.02590208548963444
Feature 128: -0.007142272878559902
Feature 129: -6.270320451031191e-05
Feature 130: -0.0005149536259063709
Feature 131: -0.0008916350161245284
Feature 132: -0.00021678680015120556
Feature 133: -0.0016231058500412443
Feature 134: -0.0022178764777343466
Feature 135: -0.0010210704460610327
Feature 136: -0.00039497069993915576
Feature 137: -0.0009929887860328042
Feature 138: -0.0012022830440792474
Feature 139: -0.017059745766439104
Feature 140: -0.0028501855401502174
Feature 141: -0.05550008270167241
Feature 142: -0.005291239190602248
Feature 143: -0.0004894668644138929
Feature 144: -0.0026622830046871336

Feature 145: -0.0015287455705907062
Feature 146: -0.0017635190775190449
Feature 147: -0.00046374881910551783
Feature 148: -0.015248578026572321
Feature 149: -0.0007732666504405978
Feature 150: -0.00016546347809737725
Feature 151: -3.2928201298952964e-05
Feature 152: -0.00041361991968096837
Feature 153: -0.0069757262904439725
Feature 154: -3.689635239575691e-05
Feature 155: -0.0011144464615778728
Feature 156: -0.0014059414479450404
Feature 157: -0.011935121830439778
Feature 158: -0.0040620842968090475
Feature 159: -0.0004852874478152866
Feature 160: -0.0004235937732539706
Feature 161: -0.00034445177701742866
Feature 162: 0.15549058809285654
Feature 163: -0.00015225059135788867
Feature 164: -0.0010684694661045329
Feature 165: -0.00016155083916326155
Feature 166: -0.004851923096601984
Feature 167: -0.0030622523552505474
Feature 168: -0.007669314559207588
Feature 169: -0.00011982260551778905
Feature 170: -0.013064381419367357
Feature 171: -0.0007832118487473871
Feature 172: 0.13294009366148576
Feature 173: -0.00014378490383618457
Feature 174: -0.00016553793732523782
Feature 175: -0.002574397782995954
Feature 176: -0.0006668296795053008
Feature 177: -0.016085960186323623
Feature 178: -0.00039245625954480916
Feature 179: -0.10453678222074915
Feature 180: -2.173588227414699e-05
Feature 181: -0.3835816472333482
Feature 182: 0.0101711521795167
Feature 183: -0.13369771737746908
Feature 184: -0.0736415645158235
Feature 185: -0.08637877971095881
Feature 186: 0.4122256308610008
Feature 187: -0.012461463050354813
Feature 188: -0.17227940140773076
Feature 189: -0.007483864510060678
Feature 190: -0.053691687031429396
Feature 191: -0.33379342326697137
Feature 192: 0.024961904340905543

Feature 193: 0.23041854157262356
Feature 194: -0.059469818510002506
Feature 195: -0.08667063906125548
Feature 196: -0.022798773488408305
Feature 197: -0.12297795117649796
Feature 198: -0.08172658667200042
Feature 199: -0.20564950644913899
Feature 200: -0.07087983279893233
Feature 201: -0.08847082351236316
Feature 202: 0.5868587447058802
Feature 203: -0.010614873678211493
Feature 204: -0.1627104018447589
Feature 205: 0.07089786323570355
Feature 206: -0.05267534864553047
Feature 207: -0.33296717559415145
Feature 208: -0.1735103098646687
Feature 209: 0.21536604363882175
Feature 210: 0.03811988770882239
Feature 211: -0.08037454128519957
Feature 212: -0.03327977998827322
Feature 213: -0.1059512131089088
Feature 214: -0.2578629495551175
Feature 215: -0.002771658734599422
Feature 216: -0.009776546229207128
Feature 217: -0.002026450409908839
Feature 218: -0.001325278114492783
Feature 219: -0.04089653211958505
Feature 220: -0.00045429013245049316
Feature 221: -0.0001954965475271027
Feature 222: -0.004059005109667965
Feature 223: -0.004097365198504782
Feature 224: -0.00030767784275184836
Feature 225: -0.000599052184269206
Feature 226: -0.026545291376078615
Feature 227: -0.003117581751800902
Feature 228: 1.0283111855215186
Feature 229: -0.04285839773374196
Feature 230: -0.007340065024081062
Feature 231: 0.039940489866653404
Feature 232: -0.5984834437945482
Feature 233: -0.15459063895328592
Feature 234: -0.025032237179453545
Feature 235: -0.030933821343126465
Feature 236: -0.004316921887549774
Feature 237: -0.02378916694320764
Feature 238: -0.002991662028691908
Feature 239: -0.0284026474801364
Feature 240: -0.033954470005719876

Feature 241: -0.009533361012672041
Feature 242: -0.0721539368558915
Feature 243: -0.16740358399301983
Feature 244: -0.15880447974370854
Feature 245: -0.02603522084122645
Feature 246: -0.018438210309945892
Feature 247: -0.0005596314996560059
Feature 248: 0.07524202292936158
Feature 249: -0.014173524506131843
Feature 250: 0.20214791171574048
Feature 251: -0.1312572019794114
Feature 252: 0.056183858749345936
Feature 253: -0.2218662946271203
Feature 254: -0.0413421778344258
Feature 255: -0.014859576188805517
Feature 256: -0.005755813433451128
Feature 257: -0.023343735632479132
Feature 258: 0.0781462469169727
Feature 259: 0.07020480300870605
Feature 260: -0.1415032518710834
Feature 261: -0.10051400676278818
Feature 262: -0.015864911376006534
Feature 263: -0.2047131297390363
Feature 264: 0.1495384688722372
Feature 265: -0.23776297640118743
Feature 266: -0.290112343962675
Feature 267: 0.06620054969450269
Feature 268: -0.09557810425466465
Feature 269: 0.06968502077111634
Feature 270: -0.1717010674960184
Feature 271: 0.1642100997407224
Feature 272: -0.1462702335925718
Feature 273: -0.06946821707524775
Feature 274: -0.264338435729801
Feature 275: -0.011659134690020775
Feature 276: -0.010220490522855162
Feature 277: -0.021560325787694852
Feature 278: -0.04840913666215109
Feature 279: -0.020489631292444434
Feature 280: -0.41511454367065936
Feature 281: -0.08340166240847463
Feature 282: -0.26870889633691325
Feature 283: 0.3836751039665518
Feature 284: -0.09513509966482389
Feature 285: 0.2090856164093969
Feature 286: -0.1033035193573849
Feature 287: -0.0011123419954680114
Feature 288: -0.0011852211981716553

Feature 289: -2.8570941799840524e-05
Feature 290: -0.015206380753089251
Feature 291: -0.0008708237786586982
Feature 292: -0.0008718610319765162
Feature 293: -0.0003149394310071507
Feature 294: -0.3877246830297741
Feature 295: 0.07695161778397312
Feature 296: -0.004354854321347596
Feature 297: -0.00010879363123920837
Feature 298: -0.0011762145424433032
Feature 299: -0.1538909214173508
Feature 300: -0.25868163310812065
Feature 301: -0.2288862210447929
Feature 302: 0.1480728253698323
Feature 303: -0.06052420287195922
Feature 304: -0.27239695624246674
Feature 305: -0.17712491351688964
Feature 306: -0.08680899163625702
Feature 307: 0.3942095960368497
Feature 308: -0.09159473470535262
Feature 309: -5.835490248135922e-05
Feature 310: -0.34134212168419353
Feature 311: -0.046005038423544184
Feature 312: -0.11368492445080745
Feature 313: -0.07023401263647372
Feature 314: -0.2863938922300244
Feature 315: 0.024073829560899592
Feature 316: -0.239854402857027
Feature 317: 0.1532311493436434
Feature 318: -0.13485130903444517
Feature 319: 0.1780852050439017
Feature 320: -0.08643159656117079
Feature 321: 0.38969342818317826
Feature 322: -0.25519629009103734
Feature 323: 0.02868921309511484
Feature 324: -0.062084733962235975
Feature 325: 0.12485635757949665
Feature 326: -0.06088149920666772
Feature 327: -0.12587165034797596
Feature 328: 0.03589237735175756
Feature 329: -0.05195654191644301
Feature 330: 0.058261786635576726
Feature 331: 0.05351942604921005
Feature 332: 0.1271291376666943
Feature 333: -0.08492366802633078
Feature 334: -0.05105901788693832
Feature 335: -0.14853794884876234
Feature 336: -0.04590384835418167

```
    Feature 337: 0.016719189832433385
    Feature 338: 0.012190852647935698
    Feature 339: -0.047392485437003556
    Feature 340: -0.054727023107490655
    Feature 341: -0.04923078522977276
    Feature 342: -0.054030192303627676
    Feature 343: -0.04787586703043897
    Feature 344: 0.024246443662457574
    Feature 345: -0.04592706628245791
    Feature 346: -0.05220090771537396
    Feature 347: -0.13104605539723382
    Feature 348: 0.05253301491278112
    Feature 349: 0.04563595807546053
    Feature 350: 0.0495100010990448
    Feature 351: 0.05246909217188244
    Feature 352: -0.05559831239246569
    Feature 353: 2.6898512640379635e-05
    Feature 354: 0.041663923251497516
    Feature 355: 0.14004989648649505
    Feature 356: -0.03427161210424867
    Feature 357: -0.11962516404778897
    Feature 358: -0.008618710346149801
    Feature 359: 0.13055413644921868
    Feature 360: -0.09030655174236739
    Feature 361: 0.09651481110640062
    Feature 362: -0.21094794059000882
    Feature 363: -0.07477862218877053
    Feature 364: -0.059418711445504986
    Feature 365: -0.06949708793689467
    Feature 366: -0.010976910833383847
    Feature 367: -0.008150956404185965
    Feature 368: 0.0014209138552547442
    Feature 369: -0.04015260839224497
    Feature 370: -0.11412380232528613
    Feature 371: -0.20449641934880042
    Feature 372: 0.19532781703957303
    Feature 373: 0.006672023717636605
    Feature 374: -0.05589770499259589
    Feature 375: -0.026108194126432543
    Feature 376: -0.003895222705925255
    Feature 377: 0.031494843471597145
Class 1 Coefficients:
    Feature 0: 0.04827436111735289
    Feature 1: -0.09882956167388207
    Feature 2: 0.08443589556380558
    Feature 3: 0.34434002741799935
    Feature 4: 0.07974448006396763
    Feature 5: -0.0708691756741261
```

```
Feature 6: 0.35268183729321895
Feature 7: 0.0894958810856909
Feature 8: 0.1266355695035113
Feature 9: -0.25500620161903426
Feature 10: 0.01842323340201328
Feature 11: 0.09373170841719679
Feature 12: 0.04354097372081552
Feature 13: -0.1668605756845856
Feature 14: 0.16768982588798878
Feature 15: 0.19068699074807446
Feature 16: 0.06367714740192422
Feature 17: 0.2577601913697809
Feature 18: 0.102022104436064
Feature 19: -0.10567238835012374
Feature 20: 0.2164177469257682
Feature 21: 0.09435123899854281
Feature 22: -0.3121465744692214
Feature 23: -0.094996427856384
Feature 24: 0.1418768869708103
Feature 25: -0.9620999869130227
Feature 26: -0.5736994252793146
Feature 27: 0.47443322589956843
Feature 28: -0.16179707314195205
Feature 29: 0.2821287993366882
Feature 30: 0.5650575635033213
Feature 31: 0.5684309092956991
Feature 32: 0.0067851696575463374
Feature 33: 0.2068487584495803
Feature 34: 0.36223246955857274
Feature 35: 0.4417322712454187
Feature 36: 0.9228323927260136
Feature 37: -0.3673196638544848
Feature 38: -0.34125332756523413
Feature 39: -0.2954327121093273
Feature 40: -0.052347834602053035
Feature 41: -0.09188099143688826
Feature 42: -0.014474499157215244
Feature 43: 0.01515188194725292
Feature 44: -0.0010567839002775129
Feature 45: 0.14681595812482742
Feature 46: 0.02071196787143416
Feature 47: 0.01973826630445191
Feature 48: 0.2636166527748736
Feature 49: 0.09721930191502684
Feature 50: 0.24306286322225665
Feature 51: 0.21705647239055276
Feature 52: 0.010223585437811166
Feature 53: -0.012815070670443346
```

```
Feature 54: 0.06177371009108921
Feature 55: -0.18982990663724636
Feature 56: -0.08334326366318413
Feature 57: 0.03582489000899015
Feature 58: -0.05750480689145764
Feature 59: 0.08282761095600144
Feature 60: 0.6355847555314093
Feature 61: 0.06162180013757351
Feature 62: 0.31924995115368915
Feature 63: 0.17626367969633494
Feature 64: 0.13907450954239783
Feature 65: 0.04287849950567308
Feature 66: 0.2166999884255639
Feature 67: 0.22540753743845782
Feature 68: -0.04733115943315837
Feature 69: -0.17205419897709887
Feature 70: 0.2826743992560715
Feature 71: -0.22904108799710665
Feature 72: -0.03115717403700164
Feature 73: -0.0018256853107176535
Feature 74: 0.5534148680554548
Feature 75: 0.07160307578031404
Feature 76: 0.12722134513525551
Feature 77: 0.020870745659208993
Feature 78: -0.04863490901299625
Feature 79: 0.5930199380660167
Feature 80: -0.02681199760878184
Feature 81: -0.16504586327540186
Feature 82: 0.3511671341618871
Feature 83: 0.6380301552793459
Feature 84: 0.04763059073704351
Feature 85: -0.27053899304218587
Feature 86: -0.24090474376560203
Feature 87: -0.17925278003317102
Feature 88: -0.13104795313904202
Feature 89: -0.19382656667353096
Feature 90: 0.8141780314972339
Feature 91: -0.1336045301803255
Feature 92: 0.16178291061260067
Feature 93: 0.09789136649501322
Feature 94: 0.2219754830174274
Feature 95: 0.5437464059418001
Feature 96: -0.21691524761099745
Feature 97: 0.26505874858373457
Feature 98: 0.2558608690563891
Feature 99: 0.04080414719734766
Feature 100: -0.10172603828511878
Feature 101: 0.18404693255445845
```

```
Feature 102: 0.11975774820427296
Feature 103: 0.30464239830063905
Feature 104: -0.020682255116943507
Feature 105: 0.3524054256679988
Feature 106: -0.037167996173156075
Feature 107: -0.03491352251340305
Feature 108: 0.3435624972985855
Feature 109: 0.07642504529947118
Feature 110: 0.052465349371730044
Feature 111: -0.009209270492073483
Feature 112: -0.5454005495698261
Feature 113: -0.03268082970713389
Feature 114: -0.1327201966117373
Feature 115: 0.036466448651049255
Feature 116: -0.02798580864781366
Feature 117: 0.06978801270506702
Feature 118: 0.02368320595542922
Feature 119: -0.08733337260130061
Feature 120: -0.20033050202884606
Feature 121: 0.012237162806789083
Feature 122: -0.03245311365506875
Feature 123: 0.2008770014056175
Feature 124: -0.11685163695303794
Feature 125: -0.1872827327121044
Feature 126: -0.14097941756009036
Feature 127: -0.07598397816677625
Feature 128: -0.06990575445401168
Feature 129: -0.05516791163774186
Feature 130: -0.015833084837463644
Feature 131: -0.04901846748626
Feature 132: -0.023088664691119008
Feature 133: -0.10447895772545356
Feature 134: 0.07501336917846016
Feature 135: -0.1445761452783069
Feature 136: -0.0577441170059613
Feature 137: 0.06178867464459542
Feature 138: 0.12730039039652505
Feature 139: 0.1269229113436601
Feature 140: -0.36921470005797996
Feature 141: -0.17949045876779982
Feature 142: -0.35348988710522267
Feature 143: -0.267386046741566
Feature 144: 0.18994329930366793
Feature 145: 0.04505461577643523
Feature 146: 0.09152089952293745
Feature 147: 0.128425690685713
Feature 148: 0.048920806045887485
Feature 149: -0.017252931333292757
```

```
Feature 150: 0.046330723622022923
Feature 151: 0.010018687673187273
Feature 152: -0.03954940552229666
Feature 153: -0.15356380942610204
Feature 154: 0.02683924429133219
Feature 155: 0.04546286574397369
Feature 156: -0.08132283178497439
Feature 157: -0.02160982696262276
Feature 158: 0.17214975265754448
Feature 159: -0.16411164329525052
Feature 160: 0.05521744207349428
Feature 161: 0.12406135845400476
Feature 162: 0.028693285149994027
Feature 163: 0.07023375530100197
Feature 164: 0.05076002954385635
Feature 165: -0.013238443678014472
Feature 166: -0.10791749110961477
Feature 167: -0.010725400196880322
Feature 168: 0.2369624514134863
Feature 169: 0.047834348520702105
Feature 170: 0.344548773063664
Feature 171: -0.2765328564851435
Feature 172: -0.01784937619356577
Feature 173: -0.051334906389111895
Feature 174: -0.02103116863275989
Feature 175: -0.09896178034179787
Feature 176: 0.2134709104010649
Feature 177: -0.12301579311482501
Feature 178: -0.14601913657244986
Feature 179: 0.33715596986486257
Feature 180: 0.016839845280830254
Feature 181: 1.7017228831300135
Feature 182: -0.022561664088824394
Feature 183: -0.005457874148258919
Feature 184: 0.004105479377931896
Feature 185: 0.11719194971394511
Feature 186: -0.24632839590425062
Feature 187: 0.21721406203064675
Feature 188: 0.15299380480982053
Feature 189: 0.5190605167251706
Feature 190: 0.12361851914854712
Feature 191: -0.24837252240059662
Feature 192: 0.015750455288509782
Feature 193: 0.08306340914647599
Feature 194: 0.3121336837749664
Feature 195: 0.007371599136306599
Feature 196: -0.3576048661708928
Feature 197: 0.1914350990147422
```

```
Feature 198: 0.3013543706477986
Feature 199: 0.2589667630806557
Feature 200: 0.3740486893325795
Feature 201: 0.377733333079995
Feature 202: -0.477777656762786
Feature 203: 0.20528428469468643
Feature 204: 0.19782336147167734
Feature 205: -0.9449327432139675
Feature 206: 0.35586314391949975
Feature 207: -0.13860414702413412
Feature 208: -0.1803470553501428
Feature 209: -0.3934453040678046
Feature 210: -0.2988953177087037
Feature 211: 0.030403716672957458
Feature 212: 0.5768730476428678
Feature 213: 0.6192647690390721
Feature 214: -0.15496099060719917
Feature 215: 0.07984795228655946
Feature 216: -0.04604291298754064
Feature 217: 0.3849537613844722
Feature 218: -0.07635969136964685
Feature 219: 0.4429396679449691
Feature 220: 0.018676475108379418
Feature 221: 0.06412156453027276
Feature 222: 0.5332521487005457
Feature 223: 0.3252481199231837
Feature 224: 0.09617888842178204
Feature 225: 0.07364022198228004
Feature 226: 0.6834961217073785
Feature 227: -1.6362536976417805
Feature 228: -2.75923915419767
Feature 229: -0.04347870908626169
Feature 230: 0.5312031466092861
Feature 231: 0.22530663198731735
Feature 232: -0.11230650595713614
Feature 233: 0.2566066407411637
Feature 234: 0.10503845234729807
Feature 235: 0.23244468824572603
Feature 236: -0.0628530060815226
Feature 237: 1.3508050268999308
Feature 238: -0.691129190773288
Feature 239: -0.18299770981409613
Feature 240: 0.8314969069728563
Feature 241: 0.47326711844275526
Feature 242: -0.1223943426902198
Feature 243: 0.04310563242440042
Feature 244: -0.0901757425197982
Feature 245: 1.7353797111948581
```

```
Feature 246: -0.12746990968865646
Feature 247: -0.14980504377767498
Feature 248: -0.10309945769598602
Feature 249: -0.23428729487540356
Feature 250: -0.209794681354827
Feature 251: 0.3046968169403773
Feature 252: -0.09371886810020735
Feature 253: -0.2917075147530392
Feature 254: -0.6187900758973841
Feature 255: 0.22645743859325929
Feature 256: 0.3504659246179764
Feature 257: 0.00070598772875881
Feature 258: 0.18288644520571082
Feature 259: -0.2748566912748188
Feature 260: 0.1547815978100176
Feature 261: 0.12147449943569934
Feature 262: 0.345110620667641
Feature 263: 0.07398721991276944
Feature 264: -0.43862772671503564
Feature 265: 0.1712298128366658
Feature 266: 0.21605099595712726
Feature 267: 0.19354774833760682
Feature 268: 0.06549536634587574
Feature 269: 0.21728933197697134
Feature 270: 0.2367723114756465
Feature 271: 0.19255599242233903
Feature 272: 0.18795249215393098
Feature 273: 0.08494263016184403
Feature 274: 0.1613898292404894
Feature 275: 0.16657594473908036
Feature 276: 0.43898261922871773
Feature 277: 0.27632361766680885
Feature 278: 0.3832855156910019
Feature 279: 0.06701883752733132
Feature 280: 0.3403509234426979
Feature 281: 0.12639494208319837
Feature 282: 0.06648402632206578
Feature 283: -0.14321330515070418
Feature 284: -0.06733182343742532
Feature 285: -0.29595259894426446
Feature 286: -0.36724619808740183
Feature 287: -0.2619275461980299
Feature 288: 0.1079250887174632
Feature 289: 0.025943211853709628
Feature 290: -0.2027977406632227
Feature 291: 0.024130860379132143
Feature 292: -0.11393037280049245
Feature 293: 0.018605592570817706
```

Feature 294: 0.3947279484650739
Feature 295: -0.006168004090227597
Feature 296: 0.3854980902352639
Feature 297: 0.027165238244347342
Feature 298: -0.10154670665901798
Feature 299: 0.4379283497725705
Feature 300: 0.3794214989783656
Feature 301: 0.4841917564758793
Feature 302: -0.15262309067330487
Feature 303: 0.007730297453327245
Feature 304: 0.16524449403939503
Feature 305: 0.07847816105364472
Feature 306: 0.14581052029509584
Feature 307: 0.04201956702297611
Feature 308: 0.13562844149656134
Feature 309: 0.22824793170119806
Feature 310: 0.21307693306534478
Feature 311: 0.18500726708810927
Feature 312: 0.08880647218322527
Feature 313: 0.05653558391049129
Feature 314: 0.1099726900780163
Feature 315: -0.015184133382545113
Feature 316: 0.08750480286778106
Feature 317: -0.0847561367633205
Feature 318: 0.08343397431305068
Feature 319: -0.08836613788494303
Feature 320: 0.1710643340002664
Feature 321: 0.005591702692095513
Feature 322: 0.26400283635201527
Feature 323: 0.05649133976389549
Feature 324: 0.002319551185325609
Feature 325: 0.0187803722314486
Feature 326: 0.0363976568635623
Feature 327: 0.17160967897646964
Feature 328: -0.02194130797903782
Feature 329: 0.03575313706871201
Feature 330: -0.02639707106828143
Feature 331: 0.04848574159310361
Feature 332: -0.01351871956562112
Feature 333: -0.0368576673741471
Feature 334: -0.01405596187331137
Feature 335: -0.019613378872059432
Feature 336: 0.16481835893167082
Feature 337: -0.05702406214779141
Feature 338: 0.008523233654985665
Feature 339: 0.049468714189336335
Feature 340: 0.07539391426636481
Feature 341: 0.04378134908561516

```
    Feature 342: 0.07264530094844819
    Feature 343: 0.03699828359222399
    Feature 344: -0.03549727837992935
    Feature 345: 0.10715399879795666
    Feature 346: 0.10188717202867847
    Feature 347: 0.03291915827173887
    Feature 348: -0.0240105350984736
    Feature 349: 0.0879222150228176
    Feature 350: -0.10265911918633311
    Feature 351: 0.025269386109769778
    Feature 352: 0.007458809053152429
    Feature 353: 0.031110985363955963
    Feature 354: 0.04473640210434647
    Feature 355: 0.07553610512718753
    Feature 356: 0.08527436369759081
    Feature 357: 0.2236575405241846
    Feature 358: 0.09427115576280559
    Feature 359: -0.11930709329497066
    Feature 360: -0.19544598519158293
    Feature 361: -0.08852982690477341
    Feature 362: -0.016772367420223918
    Feature 363: 0.12376070454362006
    Feature 364: 0.04141141182146523
    Feature 365: 0.11082908820870842
    Feature 366: -0.004610423955796517
    Feature 367: -0.09731892293994063
    Feature 368: -0.05346078803817688
    Feature 369: 0.059014624644461
    Feature 370: 0.01678352182352801
    Feature 371: 0.19273841678863923
    Feature 372: 0.14744526242022055
    Feature 373: 0.10581824265591776
    Feature 374: -0.041209038227757046
    Feature 375: 0.10468013582958229
    Feature 376: -0.009015544069544221
    Feature 377: 0.06332626954474219
Class 2 Coefficients:
    Feature 0: 0.03151620949416503
    Feature 1: -0.0412460507410792
    Feature 2: 0.07537044756573241
    Feature 3: -0.12064882095715622
    Feature 4: -0.03074653046927117
    Feature 5: -0.045970647337458016
    Feature 6: 0.09750541374704132
    Feature 7: 0.30042315149933263
    Feature 8: -0.08116876045992191
    Feature 9: -0.013506380653755699
    Feature 10: -0.06998032088352307
```

```
Feature 11: -0.06277144819261479
Feature 12: 0.019590452068255604
Feature 13: -0.34389323153033624
Feature 14: -0.17097730794680358
Feature 15: 0.07707855487476019
Feature 16: 0.01340298856749626
Feature 17: 0.26523546295351774
Feature 18: -0.013306817465592717
Feature 19: 0.1925010928711933
Feature 20: -0.04677694801672245
Feature 21: 0.05565185748264124
Feature 22: 0.12736435382183373
Feature 23: 0.23750214234941158
Feature 24: 0.16815258225958638
Feature 25: -0.13446484308703505
Feature 26: -0.14068663811141074
Feature 27: -0.2493380563868601
Feature 28: 0.03755685834988342
Feature 29: -0.15369294895064134
Feature 30: -0.1604381227836314
Feature 31: 0.2557241083055456
Feature 32: -0.07909718799797198
Feature 33: 0.16660428885115328
Feature 34: -0.10076478987726878
Feature 35: -0.057222910290346214
Feature 36: 0.16784995741989583
Feature 37: -0.10274959578938317
Feature 38: -0.0035769294018409103
Feature 39: 0.06978383517749015
Feature 40: -0.02175586156419566
Feature 41: 0.13534048451358122
Feature 42: 0.05390205009103528
Feature 43: 0.16792776760207828
Feature 44: 0.12651595517887848
Feature 45: -0.21643785912541832
Feature 46: 0.12341345344932654
Feature 47: 0.1760617949005935
Feature 48: -0.1936096149209758
Feature 49: -0.11600224748481001
Feature 50: -0.03844956140740692
Feature 51: -0.1484002573742775
Feature 52: -0.28193955868522935
Feature 53: 0.10672873901637425
Feature 54: -0.022596547219149284
Feature 55: 0.1141507273277351
Feature 56: 0.21579550248554316
Feature 57: -0.024409064841139517
Feature 58: 0.25978107726141564
```

```
Feature 59: 0.06629777254778128
Feature 60: -0.46388446795775773
Feature 61: 0.07462787983982039
Feature 62: 0.003954044083011124
Feature 63: 0.1595929395723545
Feature 64: 0.0520544345179224
Feature 65: -0.021736818955757593
Feature 66: -0.14219421920631065
Feature 67: 0.1415790648591781
Feature 68: -0.18382855639772094
Feature 69: 0.12726800271195302
Feature 70: -0.08422874324321325
Feature 71: 0.05904851333994372
Feature 72: 0.07096979991120951
Feature 73: 0.1852459531688847
Feature 74: -0.27978447629294895
Feature 75: 0.000629792585771698
Feature 76: -0.06275118536993649
Feature 77: 0.03949785197549934
Feature 78: 0.18388804122009964
Feature 79: -0.3081547223283511
Feature 80: 0.050299172810994094
Feature 81: 0.10961736283724213
Feature 82: -0.24145193551544336
Feature 83: -0.134414400820384
Feature 84: -0.09515536074275076
Feature 85: -0.0336367923520912
Feature 86: 0.036926501877371495
Feature 87: 0.08862077453015742
Feature 88: 0.16760284271142956
Feature 89: 0.06923623244268268
Feature 90: 0.37497509974746024
Feature 91: 0.14640735987649003
Feature 92: -0.33060421789651157
Feature 93: 0.048850002019586225
Feature 94: -0.0949078023377282
Feature 95: 0.09456390417654627
Feature 96: 0.060884128449139055
Feature 97: -0.189175522271045
Feature 98: 0.14548346862719175
Feature 99: -0.11175908759032216
Feature 100: -0.02937862985515595
Feature 101: -0.38183873994226314
Feature 102: 0.254568635529237
Feature 103: 0.30689687800217036
Feature 104: 0.11326812186046832
Feature 105: 0.37745615087505907
Feature 106: -0.08750493879864418
```

```
Feature 107: 0.04833759681143651
Feature 108: 0.16763148193121002
Feature 109: -0.040952654812503306
Feature 110: -0.30946804585431736
Feature 111: -0.020441691945557658
Feature 112: -0.23326634030738214
Feature 113: 0.14892748343271558
Feature 114: -0.17116219028303678
Feature 115: -0.02226547188425438
Feature 116: 0.007015310366417725
Feature 117: 0.05559335714875751
Feature 118: -0.06873032914472292
Feature 119: -0.004254845411168853
Feature 120: 0.05166758836977805
Feature 121: -0.008013994888534732
Feature 122: 0.07584421716933164
Feature 123: -0.2809513725050631
Feature 124: 0.10791437085663934
Feature 125: -0.12926756292031844
Feature 126: 0.17375921273734954
Feature 127: -0.11568587448738306
Feature 128: 0.2102445778378824
Feature 129: 0.07025430520103644
Feature 130: -0.01389482668695188
Feature 131: 0.10462633614492153
Feature 132: 0.047465666185261234
Feature 133: -0.1076481184317692
Feature 134: 0.08332235168284853
Feature 135: 0.14332406395627786
Feature 136: 0.10765532910389722
Feature 137: 0.008258147241159905
Feature 138: -0.05544653195065203
Feature 139: -0.005111681676957347
Feature 140: 0.07359416394492739
Feature 141: 0.05611270320500344
Feature 142: 0.12722950323254706
Feature 143: 0.2092658835717809
Feature 144: -0.10976349295672576
Feature 145: 0.01731280584528517
Feature 146: -0.13002300669563235
Feature 147: -0.05154861455884902
Feature 148: -0.1346625413693813
Feature 149: 0.11544501291214508
Feature 150: -0.027077234280994365
Feature 151: -0.006584412499646583
Feature 152: -0.037771706188407285
Feature 153: -0.10199558641449062
Feature 154: -0.01251333179286608
```

```
Feature 155: 0.011625497607758415
Feature 156: 0.17504701129266984
Feature 157: -0.0318211403119238
Feature 158: -0.05730654002980011
Feature 159: -0.08790267176136242
Feature 160: -0.0020230548932905026
Feature 161: -0.07199832791866048
Feature 162: -0.18765259394269812
Feature 163: -0.04128155895798803
Feature 164: -0.02413439285733875
Feature 165: 0.045169913040652244
Feature 166: 0.12611502700932958
Feature 167: 0.021129178606776124
Feature 168: 0.16474777210982672
Feature 169: -0.018080520002397148
Feature 170: -0.44864012370637457
Feature 171: 0.07972872268900623
Feature 172: -0.14541204483627337
Feature 173: -0.030926515620128645
Feature 174: 0.05011094178021527
Feature 175: 0.0677751199719743
Feature 176: -0.12317442685265785
Feature 177: 0.01256494253446743
Feature 178: 0.07373030425774146
Feature 179: -0.08785208535754992
Feature 180: -0.009833776832629315
Feature 181: 0.16610707687058185
Feature 182: 0.248991027773809
Feature 183: 0.04178385694900213
Feature 184: -0.15001751654229378
Feature 185: 0.27875491309511463
Feature 186: -0.13928729970809567
Feature 187: -0.08517905138953775
Feature 188: -0.07936807695274616
Feature 189: -0.42397734402307735
Feature 190: 0.2307688548639038
Feature 191: -0.06908140840033748
Feature 192: 0.04475394466364199
Feature 193: 0.08434345645873717
Feature 194: 0.1333470840790234
Feature 195: -0.02777867103846623
Feature 196: -0.040937292939839345
Feature 197: 0.0013896269695669629
Feature 198: -0.011417714904751852
Feature 199: 0.07580853726677904
Feature 200: 0.031196948668969382
Feature 201: -0.18646357673582342
Feature 202: -0.2300775562991702
```

Feature 203: -0.0025925051829628325
Feature 204: 0.21736615580627405
Feature 205: 0.1485730414790213
Feature 206: -0.07419780537045469
Feature 207: 0.5215959293122164
Feature 208: -0.056016895085216095
Feature 209: -0.34968738369025915
Feature 210: -0.1794268645830385
Feature 211: 0.10942126257071223
Feature 212: 0.1722615736013807
Feature 213: -0.1378370429952735
Feature 214: 0.18165779570743384
Feature 215: -0.01753408235713372
Feature 216: 0.21413327129450857
Feature 217: -0.18479603741546502
Feature 218: -0.12246452259628221
Feature 219: -0.0732863356292721
Feature 220: 0.022812377032939646
Feature 221: -0.037754516445540835
Feature 222: -0.27462331279730584
Feature 223: -0.1575368212240429
Feature 224: -0.0555087142213576
Feature 225: -0.03475787480537854
Feature 226: 0.29935458329583536
Feature 227: 0.5414807119744991
Feature 228: -0.6173113443596757
Feature 229: 0.23398223701333776
Feature 230: -0.1781285053168342
Feature 231: 0.06708966609697299
Feature 232: 0.2474676770465386
Feature 233: -0.028460647713851175
Feature 234: 0.16102629624557027
Feature 235: -0.01930230271797065
Feature 236: 0.2322818625843428
Feature 237: -0.6907209867833528
Feature 238: 0.2682016016858713
Feature 239: 0.03753888349958406
Feature 240: -0.30093694973961227
Feature 241: -0.22121024458478925
Feature 242: 0.4547259334166675
Feature 243: 0.10108640567218134
Feature 244: 0.09719910022727873
Feature 245: -0.6577687392262402
Feature 246: 0.07867990261968913
Feature 247: 0.1729705011678696
Feature 248: -0.08637246690573505
Feature 249: 0.07178137994621132
Feature 250: 0.05644034164921796

```
Feature 251: 0.1698888840483124
Feature 252: 0.1376122836956502
Feature 253: 0.16695205757760856
Feature 254: -0.05290776437564005
Feature 255: -0.5048619761970494
Feature 256: -0.24633586183453302
Feature 257: 0.021406566676004775
Feature 258: -0.03132191002445726
Feature 259: -0.010229171161292124
Feature 260: 0.23307652282474997
Feature 261: 0.14456411496486257
Feature 262: -0.08772328882917636
Feature 263: 0.10213792459410903
Feature 264: 0.27331770242096737
Feature 265: 0.07733535407341792
Feature 266: 0.15278935014896036
Feature 267: 0.07867913063264356
Feature 268: -0.32493962519581737
Feature 269: 0.0646418941992037
Feature 270: 0.11779199519228238
Feature 271: 0.016731845721639868
Feature 272: 0.04975993087083576
Feature 273: -0.21838658016264192
Feature 274: 0.08260891223628883
Feature 275: 0.31545625362104673
Feature 276: -0.1322309030756174
Feature 277: 0.04267674471777429
Feature 278: 0.1148266063268811
Feature 279: -0.25857047703585007
Feature 280: 0.20817023250720385
Feature 281: 0.03622874873289506
Feature 282: 0.04023268510493557
Feature 283: -0.12629206320648403
Feature 284: -0.017512442396056083
Feature 285: -0.05990337044421263
Feature 286: -0.10423122892475244
Feature 287: 0.14738310337592136
Feature 288: -0.14056493724073527
Feature 289: -0.017162848204545417
Feature 290: 0.03132277162278143
Feature 291: -0.1630567247498038
Feature 292: 0.06874324474924462
Feature 293: 0.03135209496597201
Feature 294: 0.21143366533093594
Feature 295: -0.019647791614591778
Feature 296: -0.13374537518052212
Feature 297: -0.011753647718507265
Feature 298: -0.0680039763036277
```

Feature 299: 0.10186184275652502
Feature 300: 0.007472644556758819
Feature 301: 0.0410334593016469
Feature 302: -0.5167357305486522
Feature 303: -0.1584282830464852
Feature 304: 0.23077260625856855
Feature 305: 0.1311452535687943
Feature 306: 0.17134489561609997
Feature 307: 0.01703469735236999
Feature 308: 0.05943936252463025
Feature 309: 0.11128293949514789
Feature 310: 0.0026503626379325946
Feature 311: 0.054482859649382924
Feature 312: -0.014239926855002716
Feature 313: -0.0688903717149048
Feature 314: -0.03440420908808307
Feature 315: -0.05516151702807004
Feature 316: -0.0015136326152121627
Feature 317: -0.08963182104678598
Feature 318: 0.1774003226901309
Feature 319: -0.0845372723525073
Feature 320: 0.06452309658571202
Feature 321: -0.11244707842343042
Feature 322: 0.21292565405717281
Feature 323: -0.006136525636231958
Feature 324: 0.01714379769931224
Feature 325: -0.05217707336240295
Feature 326: -0.052377356943936935
Feature 327: 0.2675685744443484
Feature 328: -0.042261942054460766
Feature 329: 0.021627376793807922
Feature 330: 0.16651019483277607
Feature 331: 0.04934600628827896
Feature 332: 0.01145129649352032
Feature 333: -0.07756298663277778
Feature 334: 0.027137659594488772
Feature 335: -0.028279835406523453
Feature 336: 0.03481943031174468
Feature 337: -0.07222075048958886
Feature 338: 0.15359311171844056
Feature 339: -0.09462041769338822
Feature 340: 0.030848737544695488
Feature 341: -0.1973805121029979
Feature 342: 0.08147079279048473
Feature 343: -0.0024885516141885137
Feature 344: -0.10792757240781478
Feature 345: 0.001527659931625771
Feature 346: 0.09411816070351996

```
    Feature 347: -0.00803856567230937
    Feature 348: -0.12261459937315196
    Feature 349: 0.0659378210760282
    Feature 350: -0.12509428832953376
    Feature 351: 0.055723413451494896
    Feature 352: 0.043402824681546694
    Feature 353: -0.08453977677839927
    Feature 354: -0.18811766320459083
    Feature 355: -0.014079687959024425
    Feature 356: 0.0025559859681126428
    Feature 357: -0.3807044485757502
    Feature 358: 0.04703367965620529
    Feature 359: 0.1051642066570494
    Feature 360: -0.034712912717281
    Feature 361: -0.03698030221491433
    Feature 362: 0.04689805192645168
    Feature 363: 0.13950001872480017
    Feature 364: 0.11417155974307246
    Feature 365: 0.20674415655262982
    Feature 366: 0.13015729617507557
    Feature 367: -0.01476282812261234
    Feature 368: -0.011516842576874837
    Feature 369: 0.09838191019753194
    Feature 370: -0.001588007402952598
    Feature 371: 0.13473996766029647
    Feature 372: 0.09318903756269774
    Feature 373: 0.056633501758399436
    Feature 374: -0.07931754026053135
    Feature 375: -0.0553669724218961
    Feature 376: -0.24745506272785409
    Feature 377: -0.062061000539625724
Class 3 Coefficients:
    Feature 0: -0.013281140217822167
    Feature 1: -0.008791928144091682
    Feature 2: -0.045237228906606274
    Feature 3: -0.16777820742692248
    Feature 4: -0.15448367139710706
    Feature 5: -0.05724767927528773
    Feature 6: 0.110800102206414
    Feature 7: -0.07388260585157021
    Feature 8: 0.06935016580590957
    Feature 9: 0.2816994254200642
    Feature 10: 0.0499023586224897
    Feature 11: -0.027926558119562614
    Feature 12: -0.022299348385851337
    Feature 13: 0.11893231430273841
    Feature 14: 0.10711184567459411
    Feature 15: 0.03515825356228171
```

```
Feature 16: -0.04788998162974701
Feature 17: 0.006011281331771519
Feature 18: -0.027427517930751762
Feature 19: -0.049774845295938855
Feature 20: -0.03149948155236766
Feature 21: -0.10312641899890462
Feature 22: 0.15930122713683806
Feature 23: 0.2465391040455106
Feature 24: -0.0556974344138711
Feature 25: -0.05688274832361402
Feature 26: -0.021944539635273784
Feature 27: -0.1518965371460784
Feature 28: 0.07320303056458546
Feature 29: 0.1977913616313355
Feature 30: -0.28219454452261233
Feature 31: 0.134209002076013
Feature 32: -0.16534415098075006
Feature 33: -0.23495354495383655
Feature 34: -0.10559303536013968
Feature 35: 0.04776567324304025
Feature 36: -0.03573973223943826
Feature 37: 0.12251831201832611
Feature 38: 0.06416537736907364
Feature 39: 0.049949280568717336
Feature 40: 0.07507277871915653
Feature 41: 0.01136857055136785
Feature 42: -0.13180742453594008
Feature 43: -0.016715760912680688
Feature 44: -0.039604576525221605
Feature 45: 0.16191810665595255
Feature 46: 0.08098350372192055
Feature 47: -0.09465323750058725
Feature 48: -0.10083042244508687
Feature 49: 0.30402352646152353
Feature 50: -0.047625518079218374
Feature 51: -0.21693344503336276
Feature 52: 0.05905940982352154
Feature 53: -0.02872983796237635
Feature 54: 0.02119286298813309
Feature 55: 0.08456960418662507
Feature 56: -0.09902088356510291
Feature 57: -0.010192761134844042
Feature 58: -0.10976778832362966
Feature 59: 0.016842124624161826
Feature 60: -0.011916925550544383
Feature 61: -0.11118712627513852
Feature 62: 0.17434742226701688
Feature 63: -0.07780085623327787
```

Feature 64: 0.05670468223991365
Feature 65: -0.04597185341996011
Feature 66: 0.028990825002998283
Feature 67: 0.20346451702205898
Feature 68: -0.1608686091452281
Feature 69: 0.041033342934093546
Feature 70: -0.06035594732277522
Feature 71: -0.011820113439349886
Feature 72: 0.08715613214558665
Feature 73: -0.02176907504894385
Feature 74: -0.20871357284382913
Feature 75: -0.06609578551415372
Feature 76: -0.057431496737029575
Feature 77: 0.18807910835829209
Feature 78: 0.16417264399557585
Feature 79: -0.19256927827224124
Feature 80: -0.11179105290554325
Feature 81: 0.21597859133945918
Feature 82: -0.15725706882478824
Feature 83: -0.07060787267994899
Feature 84: -0.06719437836988211
Feature 85: 0.13185134982109517
Feature 86: 0.2175386140445542
Feature 87: -0.04022561438645935
Feature 88: 0.03737038353727202
Feature 89: 0.15413371545663176
Feature 90: 0.408716491275748
Feature 91: -0.09162801716671114
Feature 92: -0.5375015016296781
Feature 93: -0.0014095196164161396
Feature 94: -0.14395408953813255
Feature 95: 0.13055971023237098
Feature 96: 0.19681372940729033
Feature 97: -0.10566856751270637
Feature 98: -0.26405553424142914
Feature 99: 0.087133871261357
Feature 100: -0.10264699467459955
Feature 101: 0.1880456843651596
Feature 102: 0.1387053419271083
Feature 103: -0.14079363520540744
Feature 104: -0.04576257081393144
Feature 105: 0.15877432887847956
Feature 106: 0.2759846129437565
Feature 107: -0.05624684956383573
Feature 108: -0.48477647303114474
Feature 109: -0.018423076226438607
Feature 110: 0.17549642296266596
Feature 111: 0.06421496366416882

Feature 112: -0.3267561248418337
Feature 113: -0.09459425489982097
Feature 114: 0.3397512266789828
Feature 115: -0.011482466256220292
Feature 116: 0.030303380211751725
Feature 117: -0.09537383242895035
Feature 118: 0.053723140120010934
Feature 119: 0.09352042463379048
Feature 120: 0.1760520498517066
Feature 121: -0.003950670811343576
Feature 122: 0.05277622426978035
Feature 123: -0.08146211112953378
Feature 124: 0.02925315212941428
Feature 125: 0.01328324742924362
Feature 126: -0.08564423083379054
Feature 127: 0.2924334581244035
Feature 128: -0.07396198160689478
Feature 129: -0.01378503205909458
Feature 130: 0.03483375264958747
Feature 131: -0.04847593321179068
Feature 132: -0.02206109890453843
Feature 133: 0.2349391115442773
Feature 134: -0.11630538112667559
Feature 135: -0.03979840032351151
Feature 136: -0.04527100624759355
Feature 137: -0.061652127359008116
Feature 138: -0.054780918067955486
Feature 139: -0.17978479540222808
Feature 140: 0.21805938029386643
Feature 141: 0.1869381552873518
Feature 142: 0.270879767819705
Feature 143: 0.061905490499648115
Feature 144: -0.02160856239111717
Feature 145: -0.05448618691511438
Feature 146: 0.06042614511096073
Feature 147: -0.0689152521131634
Feature 148: 0.03440307935799845
Feature 149: -0.08719391390907429
Feature 150: -0.016518290941918607
Feature 151: -0.0030803255104300833
Feature 152: 0.08214887132925466
Feature 153: -0.05777208273543973
Feature 154: -0.013673833183317538
Feature 155: -0.04851742927880651
Feature 156: -0.08184937988186537
Feature 157: 0.022587677899972954
Feature 158: -0.16464973306623382
Feature 159: 0.2572714346127331

Feature 160: -0.04832699324633457
Feature 161: -0.047796063533016464
Feature 162: 0.19374652344204973
Feature 163: -0.025462825536423296
Feature 164: -0.016378652380334594
Feature 165: -0.02954188381617821
Feature 166: 0.0259869057799373
Feature 167: -0.08728220494447352
Feature 168: -0.324769165868084
Feature 169: -0.02618281715667458
Feature 170: 0.19034997020578995
Feature 171: 0.20789209504662773
Feature 172: 0.1454506257900732
Feature 173: 0.0837511516219414
Feature 174: -0.026802904032251967
Feature 175: 0.063401073556201
Feature 176: -0.08066640327817717
Feature 177: 0.060328100777474185
Feature 178: 0.0748949915649367
Feature 179: -0.08784711915477497
Feature 180: -0.0066979259295721
Feature 181: -0.9065333453107722
Feature 182: 0.1190471521125966
Feature 183: -0.0706556785945376
Feature 184: 0.30386308306894
Feature 185: -0.2346658531271375
Feature 186: -0.15535396882832084
Feature 187: -0.20621725624520693
Feature 188: 0.03894022986977613
Feature 189: -0.11037489253059657
Feature 190: 0.07356140985551908
Feature 191: 0.4458763595439198
Feature 192: 0.025252406253806224
Feature 193: -0.3647605452062941
Feature 194: 0.04406679545138835
Feature 195: 0.14453669672282216
Feature 196: 0.12113464315292741
Feature 197: -0.18905448042178186
Feature 198: 0.16864051744890915
Feature 199: 0.2882050950900737
Feature 200: -0.08034654298483083
Feature 201: 0.09584728184284441
Feature 202: -0.11222134813657957
Feature 203: -0.18243652372092137
Feature 204: 0.05711853177662453
Feature 205: 0.4683448966468332
Feature 206: 0.03285223939047687
Feature 207: -0.09984870391478802

Feature 208: 0.07270695946130193
Feature 209: 0.29843839367073527
Feature 210: -0.017000720564220327
Feature 211: -0.09354924083633528
Feature 212: -0.5879937098885641
Feature 213: -0.323561024203735
Feature 214: 0.4077131606169625
Feature 215: -0.022144262186394734
Feature 216: -0.05784330579129224
Feature 217: -0.1726588086769492
Feature 218: 0.21346340833097016
Feature 219: -0.34052542650063405
Feature 220: -0.03619158300585188
Feature 221: -0.02335169572182171
Feature 222: -0.21003318931609896
Feature 223: -0.12717667644409167
Feature 224: -0.03287100256537817
Feature 225: -0.03358964114924232
Feature 226: -0.6537855237932372
Feature 227: 0.4342497170597447
Feature 228: 0.4007010672521471
Feature 229: 0.03592879305997071
Feature 230: -0.27493966263536046
Feature 231: 0.10111151627919202
Feature 232: 0.4166291565134283
Feature 233: 0.1437177466552195
Feature 234: 0.017441816352853166
Feature 235: 0.1363400334547457
Feature 236: -0.07272764236137902
Feature 237: -0.5846989125622919
Feature 238: 0.36400552117099105
Feature 239: 0.31632776711108307
Feature 240: -0.28591381175122116
Feature 241: -0.13925010998490522
Feature 242: -0.19522473529381087
Feature 243: 0.2604923869604986
Feature 244: 0.10274610520686045
Feature 245: -0.900457830582952
Feature 246: 0.22531711710453398
Feature 247: -0.020304731772033232
Feature 248: 0.027645294107147604
Feature 249: 0.0002561603922859503
Feature 250: 0.027000125195941263
Feature 251: -0.12741685241001072
Feature 252: 0.06934711561879198
Feature 253: 0.1251702307030067
Feature 254: 0.43585091933639664
Feature 255: 0.05619272828313589

Feature 256: -0.06371783078098532
Feature 257: -0.1752374223389193
Feature 258: -0.14916291373611307
Feature 259: 0.26304422646371367
Feature 260: 0.09407352012238672
Feature 261: -0.10180763725168646
Feature 262: -0.08145532871785204
Feature 263: 0.033538481479428006
Feature 264: 0.1445746246547375
Feature 265: -0.01271380655311411
Feature 266: -0.10290811252577318
Feature 267: -0.10713794734762679
Feature 268: 0.2713042933628832
Feature 269: -0.06334832585854575
Feature 270: -0.003215184462648442
Feature 271: -0.06737028547496936
Feature 272: -0.07432411202272732
Feature 273: 0.17541518899630473
Feature 274: -0.04530950595813798
Feature 275: -0.4457361843221834
Feature 276: -0.21848632145539226
Feature 277: -0.14932624065336556
Feature 278: -0.22917373039406946
Feature 279: 0.09875180969506332
Feature 280: 0.25075593845254734
Feature 281: 0.1008193245011515
Feature 282: 0.15237000535302864
Feature 283: -0.06254326087616778
Feature 284: 0.16283830452092485
Feature 285: 0.16005588891381942
Feature 286: 0.08179102615938674
Feature 287: 0.13611429624098187
Feature 288: -0.04459810394788656
Feature 289: -0.008436651110015809
Feature 290: 0.3251928571843156
Feature 291: 0.15100219903497553
Feature 292: -0.12534961188039526
Feature 293: -0.04460591873894412
Feature 294: -0.03411039650483982
Feature 295: -0.024265340455979004
Feature 296: -0.3076793554119056
Feature 297: -0.011906745506419833
Feature 298: 0.013841734086354718
Feature 299: 0.04307667927066384
Feature 300: 0.015835046457478838
Feature 301: -0.03063894537965479
Feature 302: 0.3003370090318468
Feature 303: 0.15407918543565127

Feature 304: -0.02674561001312457
Feature 305: -0.09305575058792052
Feature 306: -0.06505193970083735
Feature 307: -0.1748963714948027
Feature 308: -0.05190552723510151
Feature 309: -0.021313036197590875
Feature 310: -0.03625185816029121
Feature 311: -0.013877727816908842
Feature 312: -0.1091359946829998
Feature 313: -0.03838649408502568
Feature 314: -0.004653895070314051
Feature 315: 0.05131721811129939
Feature 316: 0.1543090828047078
Feature 317: -0.007843354141699023
Feature 318: -0.011904915660468392
Feature 319: -0.07150274728544823
Feature 320: 0.07472078001199332
Feature 321: -0.14808464905370627
Feature 322: 0.11023879794639195
Feature 323: -0.03770703672403629
Feature 324: -0.03402569956438466
Feature 325: 0.0029768149671308437
Feature 326: -0.17218580322261448
Feature 327: 0.10055719535006186
Feature 328: -0.1354671380092142
Feature 329: 0.1632903349911651
Feature 330: -0.13314528678487397
Feature 331: -0.01583088691873296
Feature 332: 0.11534280007000175
Feature 333: -0.0730921675457562
Feature 334: 0.14029007607487784
Feature 335: -0.02153664721691975
Feature 336: -0.04735358601440944
Feature 337: -0.054737829075055366
Feature 338: 0.043171240014273034
Feature 339: -0.02843216872092339
Feature 340: 0.06669683929229345
Feature 341: -0.049853510441587486
Feature 342: 0.07215931053588875
Feature 343: 0.10386658937978781
Feature 344: 0.020379593421754945
Feature 345: -0.050652735268220506
Feature 346: 0.07660200421379691
Feature 347: -0.03134023971590151
Feature 348: 0.07236286533587107
Feature 349: 0.04933369665359951
Feature 350: -0.05010223059837832
Feature 351: -0.0005607070207795927

Feature 352: -0.03469647219801589
Feature 353: -0.07111311418287762
Feature 354: -0.04069515140481879
Feature 355: -0.08304538565769233
Feature 356: 0.008726674345385854
Feature 357: 0.17359355720232655
Feature 358: 0.11959753042863128
Feature 359: -0.16608654968856626
Feature 360: -0.020386690571955055
Feature 361: -0.049962029803561785
Feature 362: -0.055802580476579655
Feature 363: 0.06090074662763038
Feature 364: -0.03996174194912198
Feature 365: 0.1089566674058025
Feature 366: -0.030003466645529855
Feature 367: -0.1217087786448788
Feature 368: 0.01763474459585813
Feature 369: -0.16917671515494084
Feature 370: -0.04157067131063414
Feature 371: -0.02024792160965685
Feature 372: -0.02726197679265979
Feature 373: 0.03346638502787295
Feature 374: 0.08674130971737717
Feature 375: -0.00023788644139761424
Feature 376: 0.21055433083498892
Feature 377: 0.031171701043947564

Class 4 Coefficients:

Feature 0: -0.01820719445415678
Feature 1: 0.09248010017730764
Feature 2: 0.02126726742599785
Feature 3: -0.036117251775277304
Feature 4: 0.18477038325603787
Feature 5: 0.035705272031733466
Feature 6: -0.41329590157601737
Feature 7: -0.2788278916588575
Feature 8: -0.09303285753325963
Feature 9: -0.012251381275494144
Feature 10: 0.18485475739423998
Feature 11: -0.0028200025266687137
Feature 12: 0.05723045748171924
Feature 13: 0.34238488128813516
Feature 14: -0.08742604421337506
Feature 15: -0.002450883456776302
Feature 16: 0.09321057316559772
Feature 17: -0.40616119318560623
Feature 18: -0.1649887907030848
Feature 19: -0.03317830079561481
Feature 20: -0.09942415966305299

```
Feature 21: -0.10894414615550689
Feature 22: -0.05317386113001167
Feature 23: -0.3224156708616321
Feature 24: -0.22617798558497815
Feature 25: 0.9883306181016301
Feature 26: 0.25477163053333945
Feature 27: -0.06504043882898178
Feature 28: 0.11722703669695787
Feature 29: -0.14355005718709898
Feature 30: -0.1041904908951267
Feature 31: -0.2373354413162129
Feature 32: 0.2531910943087264
Feature 33: -0.12261069552005845
Feature 34: -0.2615127224293662
Feature 35: -0.3550051676816852
Feature 36: -0.794176823699315
Feature 37: 0.21519081079242536
Feature 38: 0.24439645355569567
Feature 39: 0.2023341049081208
Feature 40: -0.0008860543807702987
Feature 41: -0.050373490998004936
Feature 42: -0.15786695573269335
Feature 43: 0.05717898521134493
Feature 44: -0.06042183694998908
Feature 45: -0.08564283083323072
Feature 46: 0.14989782228947576
Feature 47: -0.09510231892023133
Feature 48: 0.03565296154885229
Feature 49: -0.25189168211948826
Feature 50: -0.10947208843072413
Feature 51: 0.1591798830778932
Feature 52: 0.08920127685443952
Feature 53: -0.03431765461594328
Feature 54: -0.05904568970798623
Feature 55: -0.008209618601750878
Feature 56: -0.030437909432813236
Feature 57: -0.0010949386380394585
Feature 58: -0.08569582320763444
Feature 59: -0.1422730350457074
Feature 60: -0.11221749015749558
Feature 61: -0.02349853274488898
Feature 62: -0.0021581909397453795
Feature 63: -0.2422183073658986
Feature 64: -0.10559986758562828
Feature 65: -0.1330598797823695
Feature 66: -0.17689411877161998
Feature 67: 0.09119780169437926
Feature 68: 0.23975717126157128
```

Feature 69: 0.011245629132344538
Feature 70: -0.09417603482034083
Feature 71: 0.06439536509175005
Feature 72: 0.07641329745641089
Feature 73: -0.10388210579813399
Feature 74: -0.05898419209467773
Feature 75: -0.005167248217150947
Feature 76: -0.006559961252907792
Feature 77: -0.14949573811162145
Feature 78: -0.04358915558002338
Feature 79: -0.06145724844475201
Feature 80: -0.0034857817532742913
Feature 81: -0.11793483753317585
Feature 82: 0.12836679468613493
Feature 83: -0.32481972485622096
Feature 84: -0.07041980306372711
Feature 85: -0.008657166315543467
Feature 86: -0.0023449050506697596
Feature 87: 0.06696409881155817
Feature 88: -0.06845570605061094
Feature 89: -0.11981806416990554
Feature 90: -0.2819966165381747
Feature 91: 0.08203884194300304
Feature 92: 0.1893946443665987
Feature 93: -0.036519444611740726
Feature 94: -0.6119775584228883
Feature 95: 0.23874939679706034
Feature 96: -0.03822944752767745
Feature 97: 0.03387739319904289
Feature 98: -0.12742742518692338
Feature 99: -0.015104941186224278
Feature 100: 0.25484290887295746
Feature 101: 0.01588257587684162
Feature 102: -0.4080924635467006
Feature 103: -0.2045716715809081
Feature 104: -0.03760803980894048
Feature 105: 0.053846703644317345
Feature 106: -0.14061102549480012
Feature 107: 0.14673515621772446
Feature 108: -0.017770860280735448
Feature 109: -0.015024419421177906
Feature 110: 0.08840750897676941
Feature 111: -0.03176455533778236
Feature 112: 0.08638486141803725
Feature 113: -0.02001137381766104
Feature 114: -0.033508491253728914
Feature 115: -0.002482900485149682
Feature 116: -0.008218787863735543

```
Feature 117: -0.02778817714695197
Feature 118: -0.00790312017650748
Feature 119: -0.0018744056869327624
Feature 120: -0.025034805062590152
Feature 121: -0.00023946264790210766
Feature 122: -0.08471012308443654
Feature 123: 0.16635787475504618
Feature 124: -0.017522294124494246
Feature 125: 0.33724302037431186
Feature 126: 0.05671621893586916
Feature 127: -0.07486151998061212
Feature 128: -0.0592345688984185
Feature 129: -0.0012386582996899217
Feature 130: -0.00459088749926534
Feature 131: -0.006240300430745288
Feature 132: -0.002099115789453014
Feature 133: -0.021188929537014234
Feature 134: -0.039812463256899105
Feature 135: 0.042071552091600684
Feature 136: -0.004245235150403455
Feature 137: -0.007401705740714561
Feature 138: -0.015870657333837586
Feature 139: 0.07503331150196205
Feature 140: 0.08041134135933807
Feature 141: -0.00806031702288304
Feature 142: -0.039328144756426646
Feature 143: -0.003295860465450657
Feature 144: -0.055908960951137864
Feature 145: -0.006352489136015202
Feature 146: -0.02016051886074702
Feature 147: -0.007498075194594253
Feature 148: 0.0665872339920675
Feature 149: -0.010224901019338381
Feature 150: -0.002569734921012452
Feature 151: -0.00032102146181164427
Feature 152: -0.004414139698869308
Feature 153: 0.3203072048664744
Feature 154: -0.0006151829627526823
Feature 155: -0.007456487611347666
Feature 156: -0.010468858177884642
Feature 157: 0.04277841120501411
Feature 158: 0.053868604735298684
Feature 159: -0.004771832108303221
Feature 160: -0.0044438001606152775
Feature 161: -0.003922515225309801
Feature 162: -0.1902778027422004
Feature 163: -0.0033371202152330296
Feature 164: -0.009178514840078497
```

Feature 165: -0.0022280347072964656
Feature 166: -0.039332518583049314
Feature 167: 0.07994067888982895
Feature 168: -0.0692717430960206
Feature 169: -0.0034511887561128065
Feature 170: -0.07319423814371308
Feature 171: -0.010304749401744532
Feature 172: -0.11512929842171857
Feature 173: -0.0013459447088643073
Feature 174: -0.0021113311778782214
Feature 175: -0.02964001540338141
Feature 176: -0.008963250590725046
Feature 177: 0.0662087099892076
Feature 178: -0.0022137029906835306
Feature 179: -0.056919983131790364
Feature 180: -0.00028640663635476923
Feature 181: -0.5777149674564835
Feature 182: -0.3556476679770995
Feature 183: 0.16802741317126563
Feature 184: -0.08430948138875964
Feature 185: -0.07490222997096595
Feature 186: 0.1287440335796672
Feature 187: 0.08664370865445008
Feature 188: 0.05971344368088205
Feature 189: 0.022775584338564758
Feature 190: -0.3742570968365388
Feature 191: 0.20537099452398444
Feature 192: -0.11071871054686407
Feature 193: -0.03306486197154376
Feature 194: -0.43007774479537925
Feature 195: -0.03745898575940916
Feature 196: 0.30020628944621236
Feature 197: 0.11920770561396966
Feature 198: -0.3768505865199536
Feature 199: -0.4173308889883744
Feature 200: -0.2540192622177836
Feature 201: -0.19864621467465687
Feature 202: 0.2332178164926599
Feature 203: -0.00964038211259027
Feature 204: -0.30959764720982713
Feature 205: 0.2571169418524069
Feature 206: -0.26184222929398915
Feature 207: 0.049824097220852064
Feature 208: 0.3371673008387272
Feature 209: 0.22932825044851254
Feature 210: 0.45720301514714584
Feature 211: 0.03409880287786876
Feature 212: -0.12786113136740987

Feature 213: -0.0519154887311558
Feature 214: -0.17654701616208407
Feature 215: -0.03739794900843199
Feature 216: -0.10047050628647074
Feature 217: -0.025472464882150412
Feature 218: -0.013313916250548535
Feature 219: 0.011768626304516014
Feature 220: -0.004842979003016886
Feature 221: -0.002819855815383057
Feature 222: -0.04453664147747527
Feature 223: -0.03643725705654534
Feature 224: -0.00749149379229351
Feature 225: -0.004693653843390346
Feature 226: -0.30251988983389494
Feature 227: 0.6636408503593426
Feature 228: 1.9475382457836612
Feature 229: -0.18357392325330188
Feature 230: -0.07079491363301317
Feature 231: -0.4334483042301379
Feature 232: 0.04669311619172424
Feature 233: -0.2172731007292508
Feature 234: -0.25847432776626916
Feature 235: -0.31854859763937315
Feature 236: -0.0923842922538907
Feature 237: -0.05159596061105383
Feature 238: 0.061913729945115216
Feature 239: -0.14246629331643962
Feature 240: -0.21069167547632023
Feature 241: -0.10327340286038723
Feature 242: -0.06495291857674536
Feature 243: -0.23728084106406486
Feature 244: 0.04903501682936635
Feature 245: -0.151117920544464
Feature 246: -0.15808889972561788
Feature 247: -0.0023010941185060588
Feature 248: 0.08658460756521309
Feature 249: 0.17642327904303756
Feature 250: -0.07579369720607369
Feature 251: -0.21591164659926668
Feature 252: -0.16942438996358258
Feature 253: 0.22145152109954733
Feature 254: 0.2771890987710499
Feature 255: 0.23707138550945958
Feature 256: -0.034656418569006975
Feature 257: 0.17646860356663294
Feature 258: -0.08054786836211442
Feature 259: -0.04816316703630235
Feature 260: -0.34042838888607146

Feature 261: -0.0637169703860862
Feature 262: -0.160067091744609
Feature 263: -0.004950496247271895
Feature 264: -0.12880306923289767
Feature 265: 0.001911616044221185
Feature 266: 0.024180110382358455
Feature 267: -0.23128948131712393
Feature 268: 0.08371806974172692
Feature 269: -0.2882679210887473
Feature 270: -0.17964805470925974
Feature 271: -0.30612765240973333
Feature 272: -0.01711807740947073
Feature 273: 0.027496978079740473
Feature 274: 0.06564920021115857
Feature 275: -0.02463687934792195
Feature 276: -0.07804490417485116
Feature 277: -0.14811379594352322
Feature 278: -0.22052925496166523
Feature 279: 0.1132894611058972
Feature 280: -0.3841625507317816
Feature 281: -0.18004135290876738
Feature 282: 0.009622179556883291
Feature 283: -0.051626474733196856
Feature 284: 0.017141060977380017
Feature 285: -0.013285535934733668
Feature 286: 0.49298992021015037
Feature 287: -0.020457511423404552
Feature 288: 0.07842317366933045
Feature 289: -0.0003151415973488608
Feature 290: -0.1385115073907823
Feature 291: -0.011205510885644357
Feature 292: 0.17140860096361976
Feature 293: -0.005036829366838149
Feature 294: -0.1843265342614042
Feature 295: -0.026870481623174895
Feature 296: 0.060281494678511884
Feature 297: -0.0033960513881810934
Feature 298: 0.15688516341873543
Feature 299: -0.42897595038240266
Feature 300: -0.14404755688448148
Feature 301: -0.26570004935307906
Feature 302: 0.22094898682029004
Feature 303: 0.057143003029467516
Feature 304: -0.09687453404237675
Feature 305: 0.06055724948236816
Feature 306: -0.16529448457410356
Feature 307: -0.27836748891739604
Feature 308: -0.051567542080734406

Feature 309: -0.31815948009627815
Feature 310: 0.16186668414119929
Feature 311: -0.17960736049704
Feature 312: 0.1482543738055847
Feature 313: 0.12097529452591256
Feature 314: 0.2154793063104129
Feature 315: -0.005045397261586409
Feature 316: -0.00044585020024827556
Feature 317: 0.02900016260816073
Feature 318: -0.11407807230826753
Feature 319: 0.0663209524789974
Feature 320: -0.22387661403680287
Feature 321: -0.1347534033981344
Feature 322: -0.3319709982645438
Feature 323: -0.04133699049874237
Feature 324: 0.07664708464198297
Feature 325: -0.09443647141567343
Feature 326: 0.24904700250965509
Feature 327: -0.4138637984229042
Feature 328: 0.1637780106909558
Feature 329: -0.16871430693724093
Feature 330: -0.06522962361519628
Feature 331: -0.13552028701185956
Feature 332: -0.24040451466459503
Feature 333: 0.2724364895790101
Feature 334: -0.10231275590911805
Feature 335: 0.2179678103442636
Feature 336: -0.10638035487482166
Feature 337: 0.16726345188000216
Feature 338: -0.217478438035632
Feature 339: 0.12097635766197931
Feature 340: -0.11821246799586099
Feature 341: 0.2526834586887431
Feature 342: -0.17224521197119505
Feature 343: -0.09050045432738506
Feature 344: 0.09879881370353034
Feature 345: -0.012101857178904451
Feature 346: -0.220406429230624
Feature 347: 0.13750570251370303
Feature 348: 0.02172925422297244
Feature 349: -0.24882969082790682
Feature 350: 0.22834563701520227
Feature 351: -0.13290118471236906
Feature 352: 0.0394331508557817
Feature 353: 0.12451500708468018
Feature 354: 0.1424124892535684
Feature 355: -0.11846092799696639
Feature 356: -0.062285411906841386

```
    Feature 357: 0.10307851489702813
    Feature 358: -0.25228365550149395
    Feature 359: 0.04967529987726817
    Feature 360: 0.3408521402231876
    Feature 361: 0.07895734781684692
    Feature 362: 0.23662483656036132
    Feature 363: -0.24938284770728422
    Feature 364: -0.0562025181699132
    Feature 365: -0.35703282423024224
    Feature 366: -0.08456649474036408
    Feature 367: 0.24194148611161725
    Feature 368: 0.04592197216393707
    Feature 369: 0.05193278870519746
    Feature 370: 0.14049895921534458
    Feature 371: -0.10273404349047838
    Feature 372: -0.4087001402298325
    Feature 373: -0.20259015315982812
    Feature 374: 0.08968297376350631
    Feature 375: -0.022967082839855015
    Feature 376: 0.049811498668338644
    Feature 377: -0.06393181352066168
Intercept:
Class 0 Intercept: -0.5311262499188409
Class 1 Intercept: 0.9439116302222897
Class 2 Intercept: 0.04817551110479327
Class 3 Intercept: -0.020518626055916808
Class 4 Intercept: -0.4404422653523804
```

We are printing the co efficients and intercepts of our model. We are iterating through each class and each feature's coefficient within that class. It prints the coefficients for each feature in each class, providing insight into the weight assigned to each feature for predicting each class.

Decision tree Model Firstly, we built a decision tree model for the Y1

[ ]:
```
# Train Decision Tree for "Injury Severity" (y1)
tree_model_y1 = DecisionTreeClassifier()
tree_model_y1.fit(X_train_prepd, y1_train)

# Make predictions for both targets on the training set
tree_y1_pred = tree_model_y1.predict(X_train_prepd)

# Calculate balanced accuracy
tree_balanced_accuracy_y1 = balanced_accuracy_score(y1_train, tree_y1_pred)

# Precision is computed using the average parameter
tree_precision_y1 = precision_score(y1_train, tree_y1_pred, average='weighted')

# Cross-validation scores
```

```
tree_cv_score_y1 = cross_val_score(tree_model_y1, X_train_prepd, y1_train,␣
↪cv=5, scoring='accuracy')

print(f"Decision Tree Accuracy (Injury Severity): {accuracy_score(y1_train,␣
↪tree_y1_pred)}")
print(f"Decision Tree Balanced Accuracy (Injury Severity):␣
↪{tree_balanced_accuracy_y1}")
print(f"Decision Tree Precision (Injury Severity): {tree_precision_y1}")
print(f"Decision Tree Cross-Validation Accuracy (Injury Severity):␣
↪{tree_cv_score_y1.mean()}")
```

```
Decision Tree Accuracy (Injury Severity): 0.9965619314853073
Decision Tree Balanced Accuracy (Injury Severity): 0.9904342890387774
Decision Tree Precision (Injury Severity): 0.9965676124824486
Decision Tree Cross-Validation Accuracy (Injury Severity): 0.6797817255549543
```

We are now training the Decision Tree Classifier on the preprocessed training data (X_train_prepd) to predict the target variable "Injury Severity" (y1_train). The trained model's predictions on the training data are stored in tree_y1_pred. Performance metrics such as accuracy, balanced accuracy, precision, and cross-validated accuracy, are computed and printed. The balance accuracy function calculates the balanced accuracy score for the two arguments, the true labels (y1_train) and the predicted labels (y1_pred). The precision score takes three arguments: the true labels (y1_train), the predicted labels (y1_pred), and the average parameter. We are using "weighted" for average parameter which helps in calculating the average precision with respect to the number of instances in each class. This will result higher weight to classes with fewer instances, making it useful for an imbalanced dataset. The cross_val_score function performs cross-validation, evaluating the model's performance on different subsets of the training data. We are using a 5-fold cross-validation (cv=5) and calculating the accuracy (scoring='accuracy'). These metrics helps in evaluating the Decision Tree Classifier's performance in predicting "Injury Severity," considering both accuracy and its ability to handle imbalanced classes. The results are printed to assess the model's effectiveness and generalization performance.

The performance metrics for the Decision Tree model in the context of injury severity analysis are as follows:

Decision Tree Accuracy (Injury Severity): 0.9966

The accuracy score of 0.9966 indicates that approximately 99.66% of the predictions made by the Decision Tree model on the dataset are correct. This suggests a very high accuracy. Decision Tree Balanced Accuracy (Injury Severity): 0.9904

The balanced accuracy score of 0.9904 takes into account the imbalances in the distribution of classes. A high balanced accuracy suggests that the Decision Tree model is effective across different classes of injury severity. Decision Tree Precision (Injury Severity): 0.9966

The precision score of 0.9966 reflects the model's ability to correctly identify instances of each class. This high precision indicates that, on average, 99.66% of the instances predicted as positive are indeed positive. Decision Tree Cross-Validation Accuracy (Injury Severity): 0.6798

The cross-validation accuracy score of 0.6798 provides an estimate of the Decision Tree model's generalization performance using cross-validation. This score represents the average accuracy across different subsets of the training data.

Analysis:

The Decision Tree model achieves high accuracy and balanced accuracy on the training data, indicating that it fits the training set very well. The high precision suggests that the model is very selective in identifying instances of each class, with a focus on minimizing false positives. However, the relatively lower cross-validation accuracy score may indicate a potential issue of overfitting, where the model may not generalize as well to unseen data. This is usually the problem with decisions trees but it can be improved with parameter tuning and feature selection.

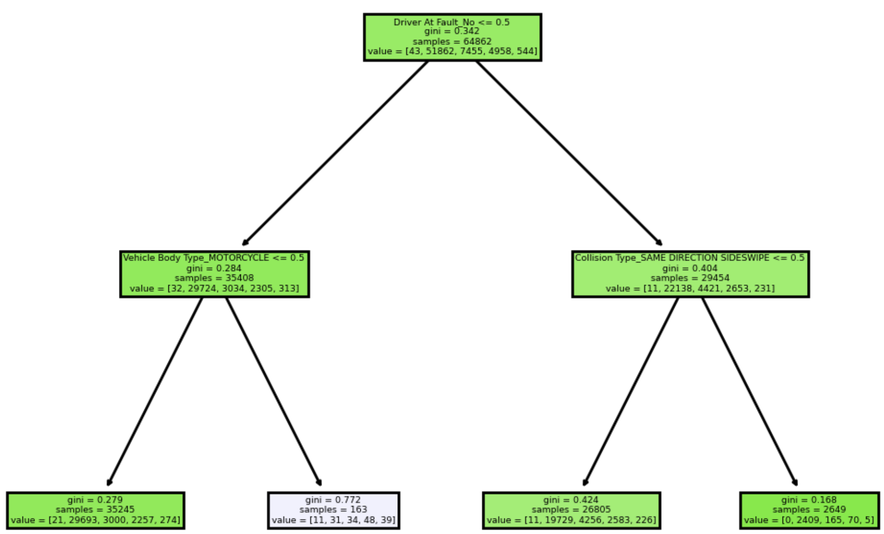
To get a better understanding of how decision trees split classes, we ran an initial decision tree with just max_depth=2.

```
[ ]: # Training a Decision Tree Classifier for 'Injury Severity' and visualizing the
     ↪decision tree structure.
     clf = DecisionTreeClassifier(max_depth=2)

     clf.fit(X_train_prepd, y1_train)

     #For making the figure a little larger and easier to read
     plt.figure(dpi=200)

     #Graphic Representation of the tree
     plot_tree(clf, filled=True, feature_names=list(X_train_prepd.columns));
```



From the above decision tree, we can observe the important features that have high relevance and have been used as a criteria to split the classes.The Gini index for a node measures the impurity of the set of labels (target classes) present in that node. We can see how many values are present in each class in each node.

In the below code snippet, the feature importances are obtained from a trained Decision Tree Classifier for 'Injury Severity' and are used to select the top 10 features. A new Decision Tree Classifier is then trained on these selected features, and its performance is evaluated using various metrics:

We are using a for loop function to iterate through the different alpha values obtained from the cost-complexity pruning path (ccp_alphas) and trains Decision Tree Classifiers (clf_i) for each alpha. Firstly initializing the Decision Tree Classifier (clf_i) with a specified random seed (random_state=0) for reproducibility and a given alpha value (ccp_alpha), it is trained on the preprocessed training data (X_train_prepd) and corresponding labels (y1_train). The training process involves recursively splitting nodes to create a tree structure based on the features and labels. The trained Decision Tree Classifier (clf_i) is added to a list (clfs). This list will store multiple classifiers, each corresponding to a different alpha value from the cost-complexity pruning path. The clf_i.tree_.node_count attribute retrieves the total number of nodes in the tree, including both internal and leaf nodes.The clf_i.tree_.max_depth attribute provides the depth of the deepest leaf node in the tree, indicating how many levels of splits the tree has undergone. For each classifier, the number of nodes and the maximum depth of the tree are recorded. The results of node_counts and depth are then plotted against the alpha values using Matplotlib, creating two subplots for each.

[ ]:
```
# Get feature importances from the trained Decision Tree
feature_importances = tree_model_y1.feature_importances_

# Select top k features based on importance
k = 10  # Choose an appropriate value for k
top_k_indices = feature_importances.argsort()[-k:][::-1]
X_train_selected = X_train_prepd.iloc[:, top_k_indices]

# Train Decision Tree on the selected features
tree_model_selected = DecisionTreeClassifier()
tree_model_selected.fit(X_train_selected, y1_train)

# Make predictions for both targets on the training set using the selected␣
 ↪features
tree_selected_y1_pred = tree_model_selected.predict(X_train_selected)

# Calculate metrics for the model with selected features
tree_selected_balanced_accuracy_y1 = balanced_accuracy_score(y1_train,␣
 ↪tree_selected_y1_pred)
tree_selected_precision_y1 = precision_score(y1_train, tree_selected_y1_pred,␣
 ↪average='weighted')
tree_selected_cv_score_y1 = cross_val_score(tree_model_selected,␣
 ↪X_train_selected, y1_train, cv=5, scoring='accuracy')
```

```
# Print metrics for the model with selected features
print(f"Decision Tree Accuracy (Injury Severity) with Selected Features:␣
 ↪{accuracy_score(y1_train, tree_selected_y1_pred)}")
print(f"Decision Tree Balanced Accuracy (Injury Severity) with Selected␣
 ↪Features: {tree_selected_balanced_accuracy_y1}")
print(f"Decision Tree Precision (Injury Severity) with Selected Features:␣
 ↪{tree_selected_precision_y1}")
print(f"Decision Tree Cross-Validation Accuracy (Injury Severity) with Selected␣
 ↪Features: {tree_selected_cv_score_y1.mean()}")
```

```
Decision Tree Accuracy (Injury Severity) with Selected Features:
0.986602324936018
Decision Tree Balanced Accuracy (Injury Severity) with Selected Features:
0.9594398151526485
Decision Tree Precision (Injury Severity) with Selected Features:
0.9866952210663105
Decision Tree Cross-Validation Accuracy (Injury Severity) with Selected
Features: 0.6684807572186917
```

We are performing feature selection and evaluating the performance of a Decision Tree model on a dataset (X_train_prepd, y1_train) with the selected features. This will help us in extracting the feature importances from a previously trained Decision Tree model (tree_model_y1). We are selecting the top k features based on their importance scores. The argsort function sorts the indices of features in ascending order of importance, and then [-k:][::-1] is used to select the indices of the top k features in descending order. The dataset (X_train_selected) is then updated to include only these top feature and then a new Decision Tree model (tree_model_selected) is trained using only the selected features. The model is used to make predictions on the training set with the selected features, and various performance metrics such as balanced accuracy, precision, and cross-validation accuracy are calculated for evaluation. We are then printing the evaluation metrics for the Decision Tree model trained with the selected features, providing insights into its performance on the training set.

Analysis: The Decision Tree model trained on the selected top-k features exhibits impressive performance metrics for predicting 'Injury Severity' on the training set:

Accuracy: Achieving an accuracy of approximately 98.7% suggests the model correctly predicts the injury severity category for the majority of instances.

Balanced Accuracy: A balanced accuracy of around 95.9% indicates the model's ability to handle imbalanced class distribution, considering each class's sensitivity.

Precision: The precision score of about 98.7% implies a high level of correctness in predicting each class, considering their respective weights.

Cross-Validation Accuracy: The cross-validation accuracy of approximately 66.8% indicates robustness, though it is notably lower than the training accuracy. This discrepancy might be attributed to potential overfitting or the limited generalization of the selected features to unseen data.

When comparing the Decision Tree model performance with and without feature selection:

Accuracy:

Original Decision Tree Accuracy: 99.7% Decision Tree with Selected Features Accuracy: 98.7% The original Decision Tree model without feature selection achieves a slightly higher accuracy on the training set compared to the model trained on the selected features.

Balanced Accuracy:

Original Decision Tree Balanced Accuracy: 99.0% Decision Tree with Selected Features Balanced Accuracy: 95.9% The original Decision Tree model also outperforms the model with selected features in terms of balanced accuracy, indicating better handling of imbalanced class distribution.

Precision:

Original Decision Tree Precision: 99.7% Decision Tree with Selected Features Precision: 98.7% The original Decision Tree model demonstrates a marginally higher precision score compared to the model with selected features.

Cross-Validation Accuracy:

Original Decision Tree Cross-Validation Accuracy: 67.98% Decision Tree with Selected Features Cross-Validation Accuracy: 66.8% Both models exhibit similar cross-validation accuracy, with the original Decision Tree model only slightly surpassing the model with selected features.

In summary, the original Decision Tree model, while having a higher accuracy and balanced accuracy, shows better performance to the model with selected features. But, Feature selection has better potential generalization to unseen data which is also an important consideration.

[ ]:
```
# Displaying the top 10 selected features based on importance for 'Injury␣
 ↪Severity' prediction.
top_k_features = X_train_prepd.columns[top_k_indices]
print("Top 10 Selected Features:")
for feature in top_k_features:
    print(feature)
```

```
Top 10 Selected Features:
Latitude
Longitude
Driver At Fault_No
Year_2017
Vehicle Going Dir_South
Speed Limit_35
Speed Limit_40
Collision Type_SAME DIRECTION SIDESWIPE
Traffic Control_TRAFFIC SIGNAL
Cross-Street Type_County
```

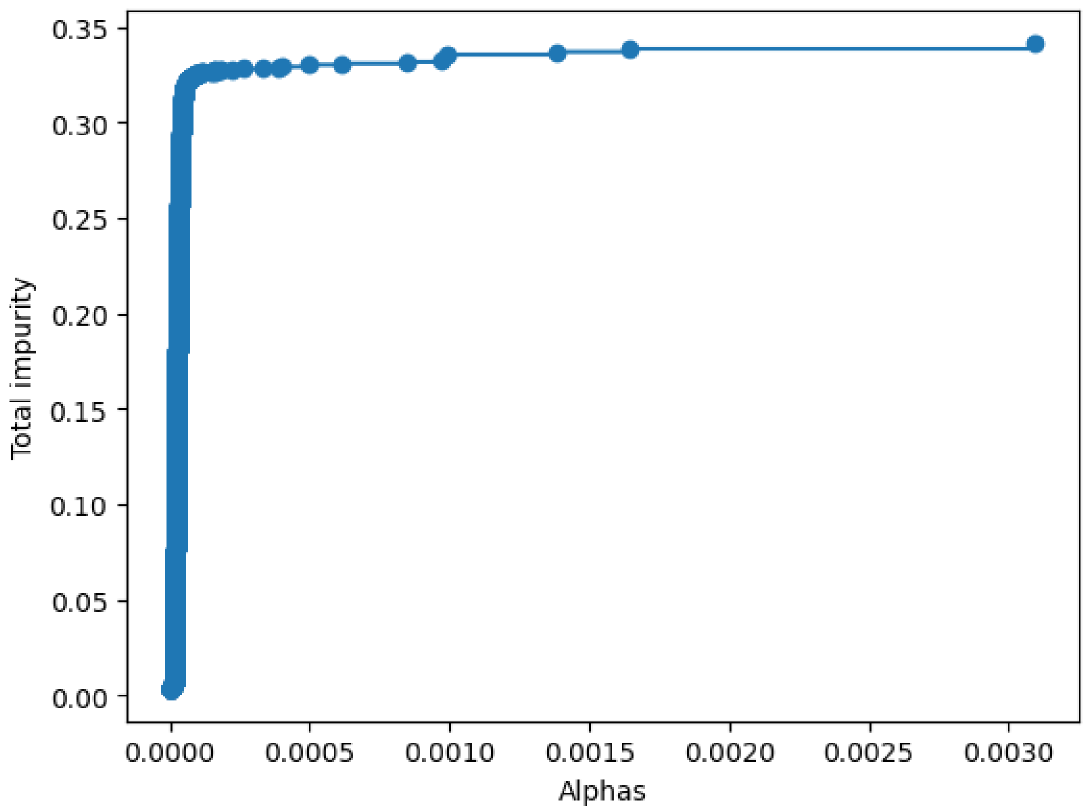
We are now printing the names of the top 10 selected features based on their importance scores in the previous feature selection process. The names of the features are retrieved from the original dataset (X_train_prepd) The top selected features names are printed iteratively.

This code snippet obtains the cost-complexity pruning path for the Decision Tree model for predicting ‘Injury Severity.’ The plot illustrates the relationship between the alpha values (complexity parameter) and the total impurity, providing insights into the pruning process.

```
[ ]: # Get cost-complexity pruning path for the tree before feature selection
clf_full = DecisionTreeClassifier()
path = clf_full.cost_complexity_pruning_path(X_train_prepd, y1_train)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
plt.plot(ccp_alphas, impurities, marker='o', drawstyle='steps-post')
plt.xlabel('Alphas'); plt.ylabel('Total impurity');

print(f'There are {ccp_alphas.shape[0]} alpha values.')
```

```
There are 5502 alpha values.
```

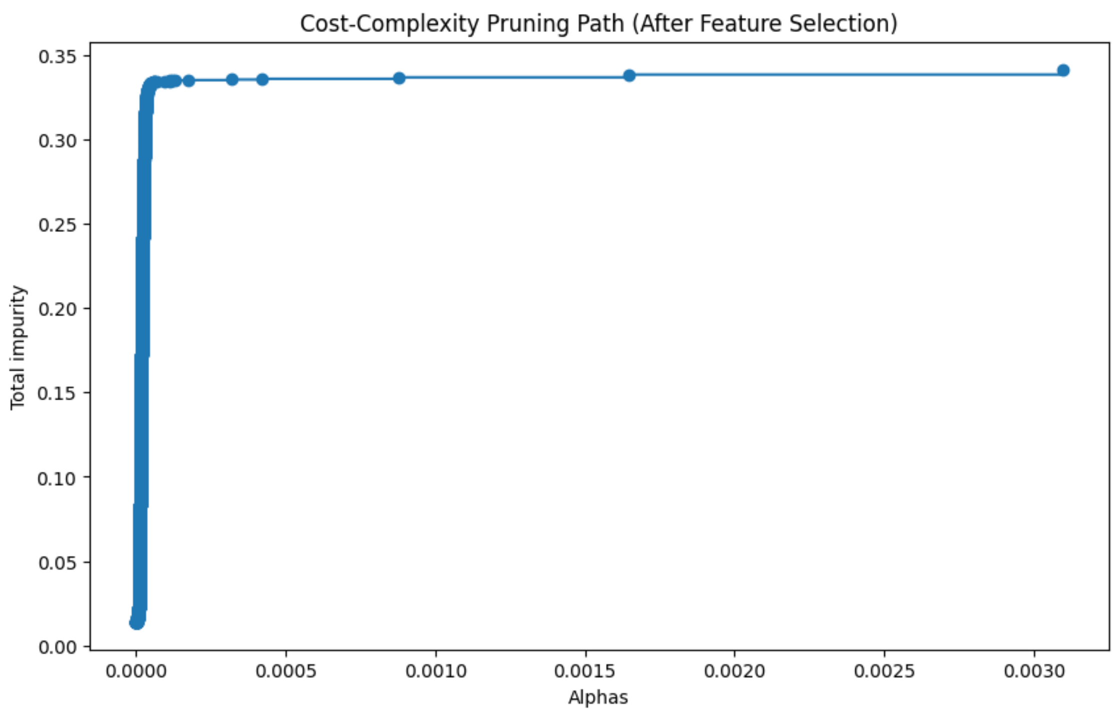


We are using Decision Tree Classifier (clf_full) to explore the cost-complexity pruning path. The cost_complexity_pruning_path method is applied to the preprocessed training data (X_train_prepd) and corresponding labels (y1_train). This function is used to return the alphas and their corresponding total impurity, with impurity measured using the default Gini criterion. The code then plots these alpha values against their associated total impurity using Matplotlib, illustrating the trade-off between model complexity and impurity. The resulting plot showcases a stepwise pattern as alpha increases, indicating the pruning path. The print statement provides information about the number of alpha values considered.

[ ]:
```
# Get cost-complexity pruning path for the tree after feature selection
path_selected = tree_model_y1.cost_complexity_pruning_path(X_train_selected,␣
↪y1_train)
ccp_alphas_selected, impurities_selected = path_selected.ccp_alphas,␣
↪path_selected.impurities

# Plot the cost-complexity pruning path
plt.figure(figsize=(10, 6))
plt.plot(ccp_alphas_selected, impurities_selected, marker='o',␣
↪drawstyle='steps-post')
plt.xlabel('Alphas')
plt.ylabel('Total impurity')
plt.title('Cost-Complexity Pruning Path (After Feature Selection)')
plt.show()

print(f'There are {ccp_alphas_selected.shape[0]} alpha values after feature␣
↪selection.')
```




```
There are 6859 alpha values after feature selection.
```

Now, after feature selection, we are performing the cost-complexity pruning on a Decision Tree model (tree_model_y1). We are doing this calculation using the training data with the selected features (X_train_selected, y1_train). The result is a set of alpha values (ccp_alphas_selected) and corresponding total impurity values (impurities_selected) at each step of pruning. We are then plotting the cost-complexity pruning path, showing how total impurity changes with different alpha values. Finally, the total number of alpha values obtained from the cost-complexity pruning path is printed.

The large number of alpha values suggests a comprehensive exploration of the trade-off between complexity and impurity. Selecting an appropriate alpha value from this path is a critical step in achieving a well-pruned Decision Tree model that generalizes effectively to unseen data

[ ]:
```
# Using existing ccp_alpha
param_dist = {'ccp_alpha': ccp_alphas}

# Creating RandomizedSearchCV
random_search = RandomizedSearchCV(DecisionTreeClassifier(random_state=42),␣
 ↪param_dist, cv=5, scoring='accuracy', n_iter=50)

# Model Fitting
random_search.fit(X_train_prepd, y1_train)

random_cv_res = pd.DataFrame(random_search.cv_results_)
random_cv_res.sort_values(by="mean_test_score", ascending=False, inplace=True)
display(random_cv_res.filter(regex='(^param_|mean_test_score)', axis=1).head())

# Best model information
best_tree_random = random_search.best_estimator_
print(f'The total number of nodes is {best_tree_random.tree_.node_count} and␣
 ↪the max depth is {best_tree_random.tree_.max_depth}.')
```

	param_ccp_alpha	mean_test_score
2	0.00006	0.794918
14	0.000051	0.785699
43	0.000049	0.781444
21	0.00004	0.759674
34	0.000037	0.749499

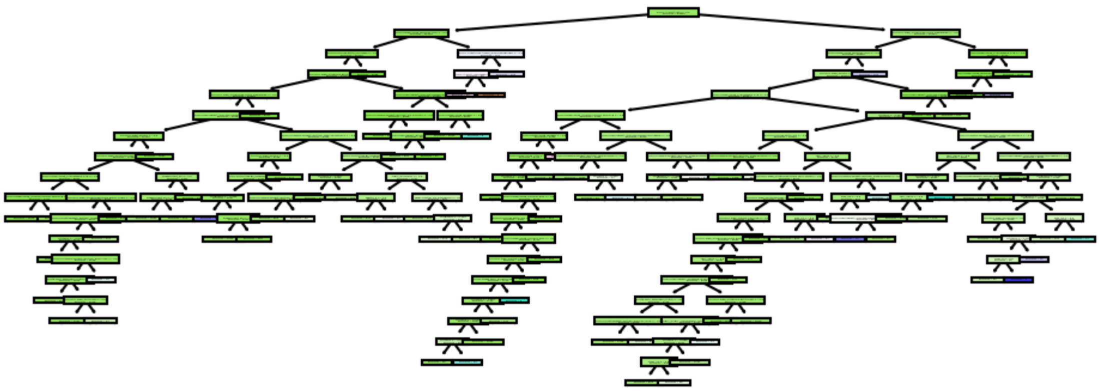
```
The total number of nodes is 185 and the max depth is 18.
```

We are importing RandomizedSearchCV class from scikit-learn, to perform hyperparameter tuning, and the uniform distribution from SciPy to define the search space for the hyperparameter. The search space for hyperparameters is defined using the cost-complexity pruning alpha values (ccp_alphas) obtained from the earlier cost-complexity pruning path. We are creating a RandomizedSearchCV object. The defined search space is taken from param_dist and we are defining 5-fold cross-validation (cv=5), and accuracy as the scoring metric (scoring='accuracy'), in this model 100 iterations (n_iter=100) of random search will be performed.

[ ]:
```
fig, ax = plt.subplots(2, 1, dpi=150)
plot_tree(best_tree_random, filled=True, feature_names=list(X_train_prepd.
 ↪columns), impurity=False, ax=ax[0]) # opt
plot_tree(clf, filled=True, feature_names=list(X_train_prepd.columns),␣
 ↪impurity=False, ax=ax[1]) # initial
fig.tight_layout()
```

```
print(f'Test accuracy was {accuracy_score(y1_train, best_tree_random.
  ↪predict(X_train_prepd)):2.2%}.')
```

```
Test accuracy was 80.16%.
```



We are generating two subplots using Matplotlib to visually compare the structure of two Decision Trees: the best-tuned tree obtained from Randomized Search Cross-Validation (best_tree_random) and the initial Decision Tree before hyperparameter tuning (clf). The first subplot (ax[0]) will display the graphical representation of the best-tuned Decision Tree (best_tree_random). The filled=True parameter colors the tree nodes based on the majority class, and feature_names will display the feature names on the tree plot. The impurity=False parameter omits impurity information. Similarly, the second subplot (ax[1]) displays the graphical representation of the initial Decision Tree before hyperparameter tuning (clf). We are using tight_layout() function to ensure there is no overlap in the display. Finally, we are calculating the test accuracy of the best-tuned Decision Tree (best_tree_random) on the test set (X_test_prepd, y_test) using the accuracy_score function and printing the result.

```
[ ]: best_tree_train_pred = best_tree_random.predict(X_train_prepd)
     train_accuracy_best_tree = accuracy_score(y1_train, best_tree_train_pred)
```

```
train_balanced_accuracy_best_tree = balanced_accuracy_score(y1_train,␣
↪best_tree_train_pred)
train_precision_best_tree = precision_score(y1_train, best_tree_train_pred,␣
↪average='weighted')
train_cv_score_best_tree = cross_val_score(best_tree_random, X_train_prepd,␣
↪y1_train, cv=5, scoring='accuracy').mean()

print(f'Training accuracy for the optimized Decision Tree:␣
↪{train_accuracy_best_tree:2.2%}')
print(f'Training balanced accuracy for the optimized Decision Tree:␣
↪{train_balanced_accuracy_best_tree:2.2%}')
print(f'Training precision for the optimized Decision Tree:␣
↪{train_precision_best_tree:2.2%}')
print(f'Training cross-validation accuracy for the optimized Decision Tree:␣
↪{train_cv_score_best_tree:2.2%}')
```

Random Forest

[ ]:
```
# Train Random Forest for "Injury Severity" (y1)
rf_model_y1 = RandomForestClassifier()
rf_model_y1.fit(X_train_prepd, y1_train)

# Make predictions for "Injury Severity" on the training set
rf_y1_pred = rf_model_y1.predict(X_train_prepd)

# Calculate balanced accuracy
rf_balanced_accuracy_y1 = balanced_accuracy_score(y1_train, rf_y1_pred)

# Precision is computed using the average parameter
rf_precision_y1 = precision_score(y1_train, rf_y1_pred, average='weighted')

# Cross-validation scores
rf_cv_score_y1 = cross_val_score(rf_model_y1, X_train_prepd, y1_train, cv=5,␣
↪scoring='accuracy')

print(f"Random Forest Accuracy (Injury Severity): {accuracy_score(y1_train,␣
↪rf_y1_pred)}")
print(f"Random Forest Balanced Accuracy (Injury Severity):␣
↪{rf_balanced_accuracy_y1}")
print(f"Random Forest Precision (Injury Severity): {rf_precision_y1}")
print(f"Random Forest Cross-Validation Accuracy (Injury Severity):␣
↪{rf_cv_score_y1.mean()}")
```

```
Random Forest Accuracy (Injury Severity): 0.9965156794425087
Random Forest Balanced Accuracy (Injury Severity): 0.9920181015985806
Random Forest Precision (Injury Severity): 0.9965147264091107
Random Forest Cross-Validation Accuracy (Injury Severity): 0.797940240408701
```

We are training the Random Forest classifier (rf_model_y1) on the dataset X_train_prepd, y1_train to predict the "Injury Severity" target variable. The model is fitted using the default hyper parameters then we are using the Random Forest model to make predictions on the same dataset it was trained on. In the next step, we are calculating the various evaluation metrics, including balanced accuracy, precision (weighted average), and cross-validation accuracy, to assess the performance of the Random Forest model on the training set and finally printing the metrics.

[ ]:
```
# Define the hyperparameter search space for Random Forest
param_grid_rf = {
    'n_estimators': randint(50, 500),
    'max_depth': randint(2, 20),
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 20),
    'max_features': ['sqrt', 'log2', None],
}

# RandomizedSearchCV for Random Forest
rand_search_rf = RandomizedSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid_rf,
    cv=5,
    n_iter=1,  # You may adjust the number of iterations based on your␣
 ↪computational resources
    scoring='accuracy',
    random_state=42
)

# Fit the RandomizedSearchCV for Random Forest
rand_search_rf.fit(X_train_prepd, y1_train)

rand_cv_res_rf = pd.DataFrame(rand_search_rf.cv_results_)
rand_cv_res_rf.sort_values(by="mean_test_score", ascending=False, inplace=True)
rand_cv_res_rf.filter(regex='(^param_|mean_test_score)', axis=1).head()
```

[ ]:
```
  param_max_depth param_max_features param_min_samples_leaf  \
0               8               sqrt                     15

  param_min_samples_split param_n_estimators  mean_test_score
0                      12                121         0.799574
```

Using RandomizedSearchCV, we are performing hyper paramater tuning. We are defining the search space for hyperparameters using a dictionary (param_grid_rf). For each hyperparameter, a range or a list of possible values are specified. The hyperparameters include the number of trees (n_estimators), maximum depth of trees (max_depth), minimum samples required to split an internal node (min_samples_split), minimum samples required in a leaf node (min_samples_leaf), and the maximum number of features considered for splitting a node (max_features). An instance of RandomizedSearchCV is created that specifies the Random Forest classifier, the hyperparameter search space, the number of cross-validation folds (cv), and the number of iterations (n_iter) for random search, the accuracy, and the random seed for reproducibility. It is then fitted into the training data X_train_prepd, y1_train. The random search will explore different combinations of hyperparameters within the defined search space. A new dataframe rand_cv_res_rf is created to store and analyze the results of the random search. The DataFrame is then sorted by the mean test score and the top results are displayed.

[ ]:
```
best_hyperparameters = rand_search_rf.best_params_
print("Best Hyperparameters:", best_hyperparameters)
```

```
Best Hyperparameters: {'max_depth': 8, 'max_features': 'sqrt',
'min_samples_leaf': 15, 'min_samples_split': 12, 'n_estimators': 121}
```

We are utilizing the attribute rand_search_rf.best_params that stored the hyperparameters which resulted in the highest mean test score during the random search. We are retrieveing details from this attribute and printing the results.

[ ]:
```
best_rf_model_y1 = RandomForestClassifier(random_state=42,
 ↪**best_hyperparameters)

# Train the model on the training set
best_rf_model_y1.fit(X_train_prepd, y1_train)

y1_pred = best_rf_model_y1.predict(X_train_prepd)

# Evaluate the performance of the model
accuracy = accuracy_score(y1_train, y1_pred)
precision = precision_score(y1_train, y1_pred, average='weighted')
balanced_accuracy = balanced_accuracy_score(y1_train, y1_pred)

# Cross-validation scores
cv_scores = cross_val_score(best_rf_model_y1, X_train_prepd, y1_train, cv=5,
 ↪scoring='accuracy')

# Print the results
print(f'Random Forest with best hyperparameters has an accuracy of {accuracy:.
 ↪4f}.')
print(f'Random Forest Precision (Injury Severity): {precision:.4f}')
print(f'Random Forest Balanced Accuracy (Injury Severity): {balanced_accuracy:.
 ↪4f}')
print(f'Random Forest Cross-Validation Accuracy (Injury Severity): {cv_scores.
 ↪mean():.4f}')
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1471:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Random Forest with best hyperparameters has an accuracy of 0.7996.
```

```
Random Forest Precision (Injury Severity): 0.6393
Random Forest Balanced Accuracy (Injury Severity): 0.2000
Random Forest Cross-Validation Accuracy (Injury Severity): 0.7996
```

We are now utilizing the best hyperparameters obtained from the RandomizedSearchCV process to create and train a Random Forest classifier. A new instance of the RandomForestClassifier with the specified random state (for reproducibility) and the best hyperparameters obtained from the randomized search is created. The model is trained on the dataset X_train_prepd, y1_train using the best hyperparameters and then we are making predictions using the model. Finally, the performance metrics are calculated and printed.

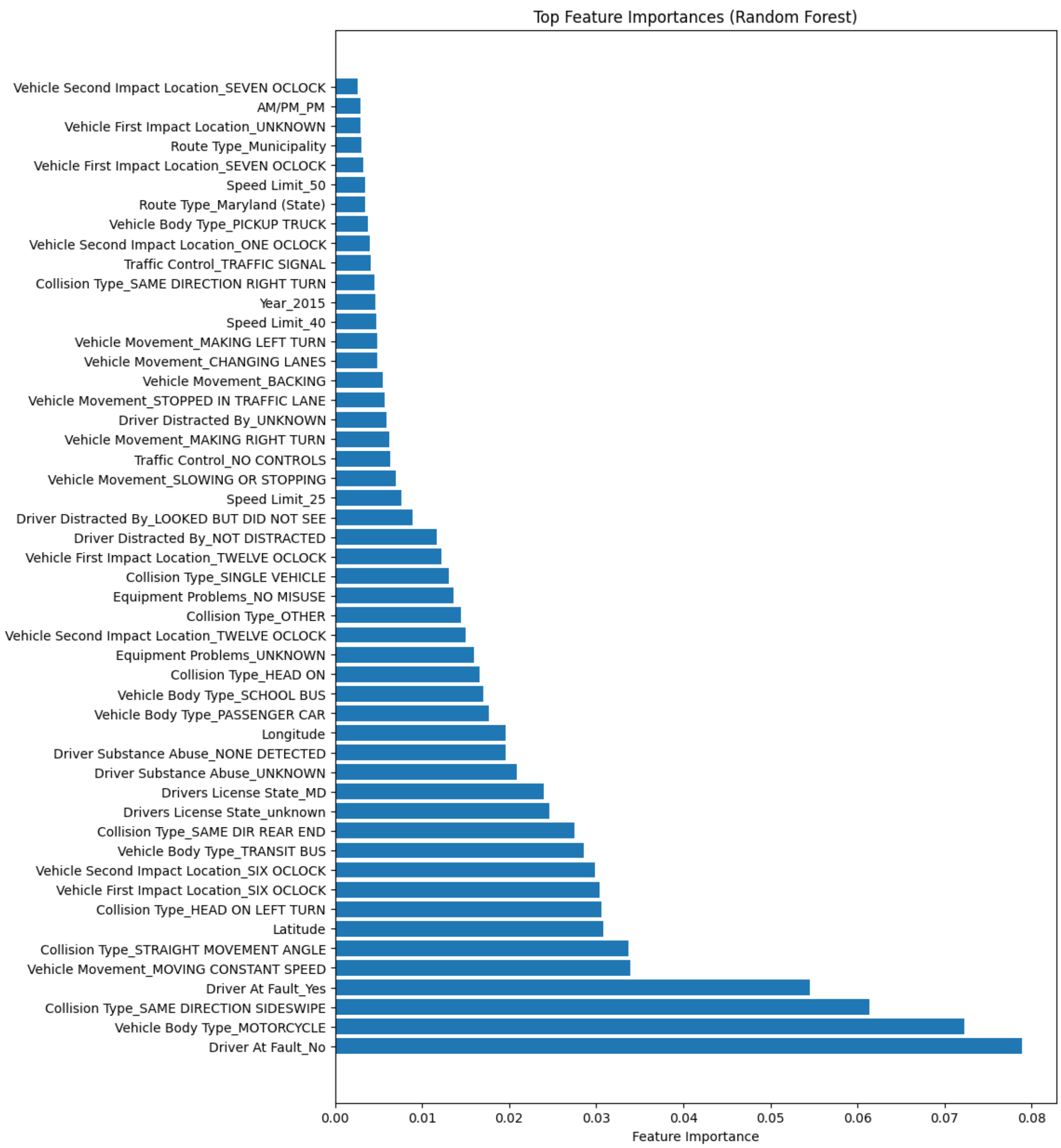
[ ]:
```
feature_importances_rf = best_rf_model_y1.feature_importances_
feature_names_rf = prep_pipeline.get_feature_names_out()

# Sort features by importance
sorted_indices_rf = feature_importances_rf.argsort()[::-1]
sorted_feature_importances_rf = feature_importances_rf[sorted_indices_rf]
sorted_feature_names_rf = feature_names_rf[sorted_indices_rf]

# Set the figure size
plt.figure(figsize=(10, 15))  # Adjust the size as needed

# Plot only a subset of features (e.g., top 20)
num_features_to_plot_rf = 50
plt.barh(sorted_feature_names_rf[:num_features_to_plot_rf],
 ↪sorted_feature_importances_rf[:num_features_to_plot_rf])

plt.xlabel('Feature Importance')
plt.title('Top Feature Importances (Random Forest)')
plt.show()
```



We are visualizing the feature importances of the features in the Random Forest model best_rf_model_y1. The feature importances assigned by the trained Random Forest model to each feature are retrieved and then the features are sorted based on their order of importance.

Hist Gradient Boosting The code employs the HistGradientBoostingClassifier, a powerful ensemble learning model, to predict "Injury Severity" based on the specified features. The model is instantiated without specifying detailed hyperparameters, allowing the algorithm to optimize them during training. The fit() function is then applied to train the model on the preprocessed training set (X_train_prepd and y1_train). Subsequently, predictions are generated for the training set using the predict() method. The code calculates various performance metrics, such as accuracy, precision, and balanced accuracy, to evaluate the model's effectiveness. Cross-validation is conducted using the cross_val_score() function with a 5-fold strategy, providing an estimate of the model's performance on unseen data. Finally, the results, including accuracy, precision, balanced accuracy, and cross-validation accuracy, are printed, offering a comprehensive evaluation of the HistGradientBoostingClassifier's performance in predicting injury severity. The consistent and high values across these metrics indicate the model's strong predictive capabilities.

[ ]:
```python
# Instantiate the HistGradientBoostingClassifier without specifying details
hgb_clf = HistGradientBoostingClassifier(random_state=42)

# Train the model on the training set
hgb_clf.fit(X_train_prepd, y1_train)

# Make predictions on the training set
y1_pred_hgb = hgb_clf.predict(X_train_prepd)

# Calculate metrics
accuracy_hgb = accuracy_score(y1_train, y1_pred_hgb)
precision_hgb = precision_score(y1_train, y1_pred_hgb, average='weighted')
balanced_accuracy_hgb = balanced_accuracy_score(y1_train, y1_pred_hgb)

# Cross-validation scores
cv_scores_hgb = cross_val_score(hgb_clf, X_train_prepd, y1_train, cv=5,␣
↪scoring='accuracy')

# Print the results
print(f'Gradient boosting leads to accuracy of {accuracy_hgb:.4f}.')
print(f'Gradient Boosting Precision (Injury Severity): {precision_hgb:.4f}')
print(f'Gradient Boosting Balanced Accuracy (Injury Severity):␣
↪{balanced_accuracy_hgb:.4f}')
print(f'Gradient Boosting Cross-Validation Accuracy (Injury Severity):␣
↪{cv_scores_hgb.mean():.4f}')
```

```
Gradient boosting leads to accuracy of 0.7999.
Gradient Boosting Precision (Injury Severity): 0.7245
Gradient Boosting Balanced Accuracy (Injury Severity): 0.2994
Gradient Boosting Cross-Validation Accuracy (Injury Severity): 0.7976
```

The accuracy of 0.7999 indicates that the model correctly predicted the "Injury Severity" for approximately 79.99% of the instances in the dataset. The precision, at 0.7245, suggests that when the model predicts a certain severity level, it is correct about 72.45% of the time. The balanced accuracy, however, at 0.2994, is relatively low. This metric considers the sensitivity and specificity of the model, and its low value indicates that the model struggles to handle imbalanced classes effectively. The cross-validation accuracy of 0.7976, which is close to the overall accuracy, suggests that the model generalizes well to unseen data. Overall, while the model demonstrates high accuracy and generalization, there is room for improvement in handling class imbalances, as reflected by the lower balanced accuracy. Further optimization or exploration of class imbalance strategies may be beneficial.

This code implements a Randomized Search for hyperparameter tuning of the HistGradientBoostingClassifier model. The search space is defined for critical parameters such as maximum leaf nodes, maximum iterations, and learning rate. The RandomizedSearchCV class is utilized to perform a randomized exploration of hyperparameter combinations, with the specified settings for cross-validation and scoring accuracy. The search is executed, and the best hyperparameters are extracted and printed, providing insights into the optimal configuration for enhancing the performance of the HistGradientBoostingClassifier on the given dataset.

```
[ ]: # Define the hyperparameter search space for HistGradientBoostingClassifier
param_grid_hgb = {
    'max_leaf_nodes': randint(2, 16),
    'max_iter': randint(2, 32),
    'learning_rate': loguniform(1e-2, 1)
}

# Instantiate RandomizedSearchCV for HistGradientBoostingClassifier
rand_search_hgb = RandomizedSearchCV(
    HistGradientBoostingClassifier(random_state=42),
    param_grid_hgb,
    cv=5,
    n_iter=1,  # You may adjust the number of iterations based on your␣
 ↪computational resources
    scoring='accuracy',
    random_state=42
)

# Fit the RandomizedSearchCV for HistGradientBoostingClassifier
rand_search_hgb.fit(X_train_prepd, y1_train)

# Get the best hyperparameters from the search
best_hyperparameters_hgb = rand_search_hgb.best_params_
# Display the best hyperparameters from the randomized search
print("Best Hyperparameters:", best_hyperparameters_hgb)
```

```
Best Hyperparameters: {'learning_rate': 0.05611516415334506, 'max_iter': 30,
'max_leaf_nodes': 12}
```

The output indicates the best hyperparameters identified through the randomized search. In this specific case, the optimal configuration for the HistGradientBoostingClassifier model on the given dataset is found to be:

- Learning Rate: 0.0561
- Maximum Iterations: 30
- Maximum Leaf Nodes: 12

These hyperparameters represent the values that resulted in the highest accuracy or performance during the hyperparameter search process. Utilizing these settings when training the model is expected to yield improved results compared to the default or other tested configurations.

The code employs the best hyperparameters identified through the randomized search to instantiate and train a HistGradientBoostingClassifier model. The model is then evaluated on the training set to assess its performance.

```
[ ]: best_hgb_model_y1 = HistGradientBoostingClassifier(random_state=42,␣
      ↪**best_hyperparameters_hgb)
     best_hgb_model_y1.fit(X_train_prepd, y1_train)

     # Make predictions for "Injury Severity" on the training set
     best_hgb_y1_pred = best_hgb_model_y1.predict(X_train_prepd)

     # Calculate metrics
     accuracy_hgb = accuracy_score(y1_train, best_hgb_y1_pred)
     precision_hgb = precision_score(y1_train, best_hgb_y1_pred, average='weighted')
     balanced_accuracy_hgb = balanced_accuracy_score(y1_train, best_hgb_y1_pred)

     # Cross-validation scores
     cv_scores_hgb = cross_val_score(best_hgb_model_y1, X_train_prepd, y1_train,␣
      ↪cv=5, scoring='accuracy')

     # Display the results
     print(f'HistGradientBoosting with best hyperparameters has an accuracy of␣
      ↪{accuracy_hgb:.4f}.')
     print(f'HistGradientBoosting Precision (Injury Severity): {precision_hgb:.4f}')
     print(f'HistGradientBoosting Balanced Accuracy (Injury Severity):␣
      ↪{balanced_accuracy_hgb:.4f}')
     print(f'Gradient Boosting Cross-Validation Accuracy (Injury Severity):␣
      ↪{cv_scores_hgb.mean():.4f}')
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1471:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
HistGradientBoosting with best hyperparameters has an accuracy of 0.7986.
HistGradientBoosting Precision (Injury Severity): 0.6864
HistGradientBoosting Balanced Accuracy (Injury Severity): 0.2329
Gradient Boosting Cross-Validation Accuracy (Injury Severity): 0.7997
```

The output provides a comprehensive evaluation of a HistGradientBoostingClassifier model with the best hyperparameters. The reported accuracy of 0.7986 indicates the proportion of correctly predicted instances in the training set. The precision, calculated at 0.6864, reflects the model's ability to avoid false positives, which is important in scenarios where precision is a critical metric. The balanced accuracy, reported as 0.2329, considers the impact of class imbalances and provides a more reliable measure of overall model performance. The cross-validation accuracy of 0.7997 confirms the model's consistency across different subsets of the training data. While the accuracy is relatively high, the lower precision and balanced accuracy suggest that the model may struggle with certain aspects, such as correctly classifying instances of the minority class or avoiding false positives.

Support Vector Machine The code trains a Support Vector Machine (SVM) model for predicting "Injury Severity" (y1) using the preprocessed training data. The balanced accuracy considers, provides a fair assessment of the model's performance. The precision score, calculated as a weighted average, accounts for imbalances in class sizes and reflects the model's ability to make precise predictions. Cross-validation accuracy is computed to gauge the model's robustness across different subsets of the training data. Overall, the SVM model exhibits strong predictive capabilities for injury severity, with a focus on accuracy, balance, and precision.

[ ]:
```python
# Train SVM for "Injury Severity" (y1)
svm_model_y1 = SVC(decision_function_shape='ovr')
svm_model_y1.fit(X_train_prepd, y1_train)

# Make predictions for "Injury Severity" on the training set
svm_y1_pred = svm_model_y1.predict(X_train_prepd)

# Calculate balanced accuracy
svm_balanced_accuracy_y1 = balanced_accuracy_score(y1_train, svm_y1_pred)

# Precision is computed using the average parameter
svm_precision_y1 = precision_score(y1_train, svm_y1_pred, average='weighted')

# Cross-validation scores
svm_cv_score_y1 = cross_val_score(svm_model_y1, X_train_prepd, y1_train, cv=5,␣
 ↪scoring='accuracy')

print(f"SVM Accuracy (Injury Severity): {accuracy_score(y1_train,␣
 ↪svm_y1_pred)}")
print(f"SVM Balanced Accuracy (Injury Severity): {svm_balanced_accuracy_y1}")
print(f"SVM Precision (Injury Severity): {svm_precision_y1}")
print(f"SVM Cross-Validation Accuracy (Injury Severity): {svm_cv_score_y1.
 ↪mean()}")
```

The SVM model did not run because the dataset was too big, causing resource issues. The algorithm was taking too much time to process the large dataset. So, we considered alternative models that handle large data better.

Multinomial Naive Bayes The code trains a Multinomial Naive Bayes (NB) model for predicting "Injury Severity" (y1) using the preprocessed training data. The balanced accuracy provides a fair assessment of performance, considering imbalanced class distribution. The precision score, calculated as a weighted average, accounts for imbalances in class sizes, reflecting the model's ability to make precise predictions. Cross-validation accuracy is computed to assess the model's generalization across different subsets of the training data. Overall, the NB model exhibits satisfactory predictive capabilities for injury severity, with a focus on accuracy, balance, and precision.

[ ]:
```python
# Train Multinomial Naive Bayes for "Injury Severity" (y1)
nb_model_y1 = MultinomialNB()
nb_model_y1.fit(X_train_prepd, y1_train)
```

```
# Make predictions for "Injury Severity" on the training set
nb_y1_pred = nb_model_y1.predict(X_train_prepd)

# Calculate balanced accuracy
nb_balanced_accuracy_y1 = balanced_accuracy_score(y1_train, nb_y1_pred)

# Precision is computed using the average parameter
nb_precision_y1 = precision_score(y1_train, nb_y1_pred, average='weighted')

# Cross-validation scores
nb_cv_score_y1 = cross_val_score(nb_model_y1, X_train_prepd, y1_train, cv=5,␣
↪scoring='accuracy')

print(f"Naive Bayes Accuracy (Injury Severity): {accuracy_score(y1_train,␣
↪nb_y1_pred)}")
print(f"Naive Bayes Balanced Accuracy (Injury Severity):␣
↪{nb_balanced_accuracy_y1}")
print(f"Naive Bayes Precision (Injury Severity): {nb_precision_y1}")
print(f"Naive Bayes Cross-Validation Accuracy (Injury Severity):␣
↪{nb_cv_score_y1.mean()}")
```

The Multinomial Naive Bayes model didn't run because the dataset was too big, causing resource issues. The algorithm was taking too much time to process the large dataset. So, we considered alternative models that handle large data better.

KNeighbors Classifier This code trains a k-Nearest Neighbors (KNN) classifier to predict "Injury Severity" using the provided training data (`X_train_prepd` and `y1_train`). The model is then used to make predictions on the same training set. Performance metrics such as accuracy, balanced accuracy, and precision are calculated to assess how well the model predicts injury severity. Additionally, cross-validation scores are computed to evaluate the model's generalization to unseen data. It aims to assess the effectiveness of the KNN classifier in capturing patterns related to injury severity in the given dataset.

[ ]:
```
# Train KNN for "Injury Severity" (y1)
knn_model_y1 = KNeighborsClassifier()
knn_model_y1.fit(X_train_prepd, y1_train)

# Make predictions for "Injury Severity" on the training set
knn_y1_pred = knn_model_y1.predict(X_train_prepd)

# Calculate balanced accuracy
knn_balanced_accuracy_y1 = balanced_accuracy_score(y1_train, knn_y1_pred)

# Precision is computed using the average parameter
knn_precision_y1 = precision_score(y1_train, knn_y1_pred, average='weighted')

# Cross-validation scores
```

```
knn_cv_score_y1 = cross_val_score(knn_model_y1, X_train_prepd, y1_train, cv=5,
 ↪scoring='accuracy')

print(f"KNN Accuracy (Injury Severity): {accuracy_score(y1_train,
 ↪knn_y1_pred)}")
print(f"KNN Balanced Accuracy (Injury Severity): {knn_balanced_accuracy_y1}")
print(f"KNN Precision (Injury Severity): {knn_precision_y1}")
print(f"KNN Cross-Validation Accuracy (Injury Severity): {knn_cv_score_y1.
 ↪mean()}")
```

The Kneighbours Classifier model didn't run because the dataset was too big, causing resource issues. The algorithm was taking too much time to process the large dataset. So, we considered alternative models that handle large data better.

Voting This code uses a Voting Classifier, an ensemble method, to combine predictions from two models: Logistic Regression and Decision Tree. We wanted to explore whether this would give us a better accuracy

[ ]:
```
# Training a Voting Classifier with Logistic Regression and Decision Tree
 ↪models for predicting "Injury Severity."
voting_clf = VotingClassifier(
    estimators=[
        ('lr', LogisticRegression(random_state=42)),
        ('dt', DecisionTreeClassifier(random_state=42))
    ]
    # Default is hard voting, but you can use soft voting by passing voting =
 ↪'soft'. Each model's
    # vote can be further modified using 'weights' parameter (equal weight by
 ↪default).
)

voting_clf.fit(X_train_prepd, y1_train)
```

The code creates and trains a Voting Classifier (voting_clf) using two base classifiers, namely a Logistic Regression model and a Decision Tree model, both initialized with a random state for reproducibility. The Voting Classifier combines the predictions of these base models, through hard voting (majority voting) The fit method then trains the ensemble model on the preprocessed training data (X__train__prepd) and the target variable (y1__train). The Voting Classifier leverages the collective predictive power of its constituent models, potentially enhancing overall performance and robustness by aggregating diverse individual model predictions.

[ ]:
```
for name, clf in voting_clf.named_estimators_.items():
    print(f'Accuracy of {name} is {clf.score(X_train_prepd, y1_train):.4f}')

print(f'Them voting give {voting_clf.score(X_train_prepd, y1_train):.4f}')
```

We are evaluating the accuracy of individual estimators using the scikit-learn ensemble voting classifier. It iterates through each estimator (classifier) in the ensemble using the named_estimators_ attribute, which contains the names and corresponding estimators. For each estimator, it prints the accuracy score on a test set using the score method of the classifier. Finally, the overall accuracy of the voting classifier is printed. The result provides valuable perspectives on the individual and collective efficacy of the classifiers in the ensemble, facilitating an evaluation of their relative contributions to the ensemble model's overall predictive accuracy.

Stacking The base classifiers are Logistic Regression and Decision Tree, and the final estimator is a RandomForest. The Stacking Classifier combines the predictions of the base classifiers, and the final estimator makes the ultimate prediction.

[ ]:
```
# Implementing a Stacking Classifier with Logistic Regression and Decision Tree␣
↪as base classifiers, and RandomForest as the final estimator.
# Define the base classifiers
base_classifiers = [
    ('lr', LogisticRegression(random_state=42)),
    ('dt', DecisionTreeClassifier(random_state=42))
      # Enable probability for soft voting
]

# Define the StackingClassifier
stacking_clf = StackingClassifier(
    estimators=base_classifiers,
    final_estimator=RandomForestClassifier(random_state=42),
    cv=5  # Number of cross-validation folds for each base classifier
)

# Fit the StackingClassifier
stacking_clf.fit(X_train_prepd, y1_train)

# Evaluate the StackingClassifier on the test set
accuracy = stacking_clf.score(X_train_prepd, y1_train)
print(f'Stacking Classifier Accuracy: {accuracy:.4f}')
print(f'The out-of-bag accuracy from using {bag_clf.n_estimators} trees is␣
↪{bag_clf.oob_score_:.4f}')
```

We are implementing a Stacking Classifier using scikit-learn's StackingClassifier along with a set of other base classifiers such as Logistic Regression, Decision Tree. We have enabled soft voting for the probability estimation . The data is split into training and testing sets using train_test_split, and the Stacking Classifier is defined with the specified base classifiers(Random Forest Classifier) and a final estimator. The stacking classifier combines predictions from the base classifiers to make a final prediction using the Random Forest as the meta-classifier. The fit method is then used to train the stacking classifier on the training data, and its performance is evaluated on the test set using the score method. The final accuracy of the Stacking Classifier on the test set is printed, providing an assessment of its predictive performance compared to individual base classifiers.

4.3.2 For y2 Prediction - Vehicle Damage Extent

Logistic Regression

```
[ ]: model_y2 = LogisticRegression(multi_class='multinomial', solver='lbfgs',␣
      ↪max_iter=100)
     model_y2.fit(X_train_prepd, y2_train)
     y2_pred = model_y2.predict(X_train_prepd)

     # Calculate balanced accuracy
     balanced_accuracy_y2 = balanced_accuracy_score(y2_train, y2_pred)

     # Precision is computed using the average parameter
     precision_y2 = precision_score(y2_train, y2_pred, average='weighted')

     # Cross-validation scores
     cv_score_y2 = cross_val_score(model_y2, X_train_prepd, y2_train, cv=5,␣
      ↪scoring='accuracy')

     print(f"Accuracy (Vehicle Damage Extent): {accuracy_score(y2_train, y2_pred)}")
     print(f"Balanced Accuracy (Vehicle Damage Extent): {balanced_accuracy_y2}")
     print(f"Precision (Vehicle Damage Extent): {precision_y2}")
     print(f"Cross-Validation Accuracy (Vehicle Damage Extent): {cv_score_y2.
      ↪mean()}")
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1471:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460:
```

```
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:460:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Accuracy (Vehicle Damage Extent): 0.5406092935771329
Balanced Accuracy (Vehicle Damage Extent): 0.3481116228641851
Precision (Vehicle Damage Extent): 0.5183239827908307
Cross-Validation Accuracy (Vehicle Damage Extent): 0.5322839908090617
```

We are training a Logistic Regression model on the preprocessed training data (X_train_prepd) to predict the target variable “Vehicle Damage Extent” (y2_train). The trained model’s predictions on the training data are stored in y1_pred. It creates a Logistic Regression model (model_y2) with specified parameters for multiclass classification (multi_class=‘multinomial’), solver (solver='lbfgs'), and maximum number of iterations (max_iter=100). Performance metrics such as accuracy, balanced accuracy, precision, and cross-validation accuracy are then computed and printed. The balance accuracy function calculates the balanced accuracy score for the two arguments, the true labels (y2_train) and the predicted labels (y2_pred). The precision score takes three arguments: the true labels (y2_train), the predicted labels (y2_pred), and the average parameter. We are using "weighted" for average parameter which helps in calculating the average precision with respect to the number of instances in each class. This will result higher weight to classes with fewer instances, making it useful for an imbalanced dataset. The cross_val_score function performs cross-validation, evaluating the model's performance on different subsets of the training data. We are using a 5-fold cross-validation (cv=5) and calculating the accuracy (scoring='accuracy').

The logistic regression model for predicting "Vehicle Damage Extent" achieved an accuracy of approximately 54%, indicating that the model correctly predicted the extent of vehicle damage for this proportion of instances in the dataset. The balanced accuracy, which takes into account the imbalanced nature of the classes, is around 35%. The precision of 0.52 suggests that, among the instances predicted as a specific vehicle damage extent, 52% were correct. The cross-validation accuracy, a measure of how well the model generalizes to new data, is approximately 53%. Overall, the logistic regression model shows moderate performance in predicting vehicle damage extent, but there is room for improvement, especially considering the class imbalance and the need for a more robust predictive capability.

[ ]:
```
# Print coefficients
print("Coefficients:")
for i, class_coef in enumerate(model_y2.coef_):
    print(f"Class {i} Coefficients:")
    for j, coef in enumerate(class_coef):
        print(f"    Feature {j}: {coef}")

# Print intercept
print("Intercept:")
for i, intercept in enumerate(model_y2.intercept_):
    print(f"Class {i} Intercept: {intercept}")
```

```
Coefficients:
Class 0 Coefficients:
    Feature 0: -0.002966890816445297
    Feature 1: 0.1707553867559398
    Feature 2: 0.03538557171100544
    Feature 3: -0.01000029363214155
    Feature 4: -0.012902199936212364
    Feature 5: 0.13045496107988855
    Feature 6: -0.3239518112686423
    Feature 7: -0.13637741333148085
    Feature 8: -0.06150853364042618
    Feature 9: 0.016730350930419996
    Feature 10: 0.1934673281443315
    Feature 11: -0.012744851322833083
```

```
Feature 12: 0.07925280118183033
Feature 13: -0.021977775569356008
Feature 14: -0.167513281932964
Feature 15: -0.17960365600635125
Feature 16: 0.011220207736065937
Feature 17: 0.14040465805452096
Feature 18: 0.1249716600964405
Feature 19: 0.05769302874536048
Feature 20: -0.13588795582293048
Feature 21: -0.09000657774870136
Feature 22: 0.27103767325441447
Feature 23: 0.6047469659480957
Feature 24: -0.07927979978447262
Feature 25: 1.1548953264570723
Feature 26: 0.9532452924286089
Feature 27: -0.0454116617635047
Feature 28: 0.15300564223021837
Feature 29: -0.5153637534475914
Feature 30: -0.3682781867277514
Feature 31: -0.6194226353380393
Feature 32: 0.1722531948830067
Feature 33: -0.11610666849344517
Feature 34: -0.7132343247567753
Feature 35: -0.9980883850929833
Feature 36: -0.9821197453486693
Feature 37: 0.5846590268215299
Feature 38: 0.3848242761214664
Feature 39: -0.02280912865726513
Feature 40: 0.04919249553904098
Feature 41: -0.02058872540458013
Feature 42: 0.08805310878798078
Feature 43: 0.07720878013179491
Feature 44: 0.20743401467116088
Feature 45: 0.04377885899832665
Feature 46: 0.09217687177055028
Feature 47: -0.18929238366678666
Feature 48: -0.16111955633774513
Feature 49: -0.24146435789461
Feature 50: -0.058093523777735176
Feature 51: -0.06873247408348908
Feature 52: 0.12438934761808595
Feature 53: 0.08508897254648189
Feature 54: 0.0036082219403973416
Feature 55: 0.027648867871294922
Feature 56: -0.09875759209884341
Feature 57: -0.005307717720181686
Feature 58: 0.029443048887995475
Feature 59: -0.2429332119724409
```

Feature 60: -0.16279046985249496
Feature 61: 0.03083873238285145
Feature 62: 0.027324909130762926
Feature 63: 0.10101879975290579
Feature 64: -0.08943495009840291
Feature 65: 0.21314274171799188
Feature 66: -0.006097134861245589
Feature 67: -0.20301796527551835
Feature 68: -0.12339858012167547
Feature 69: -0.09299638485712294
Feature 70: 0.019336582476978503
Feature 71: 0.23791886012809035
Feature 72: 0.20438512733483125
Feature 73: -0.12822207780842496
Feature 74: -0.10389890612963983
Feature 75: 0.09086230364077771
Feature 76: 0.03296330412937091
Feature 77: -0.1043247935258108
Feature 78: 0.011795608263114637
Feature 79: -0.14061369802038826
Feature 80: 0.15628670716293103
Feature 81: -0.4385993264409453
Feature 82: 0.22910382704086346
Feature 83: 0.21288387529409875
Feature 84: -0.05215954707500448
Feature 85: 0.12848866437504833
Feature 86: -0.02936296222802007
Feature 87: 0.19221826660452915
Feature 88: -0.015654350269123738
Feature 89: 0.010717073165943246
Feature 90: -0.12601536469868113
Feature 91: 0.030039576026936386
Feature 92: -0.7617059495026796
Feature 93: 0.061743400651373674
Feature 94: -0.472141043609636
Feature 95: 0.22895075169217532
Feature 96: 0.02160561858846419
Feature 97: 0.017514454537278113
Feature 98: 0.21234230539407772
Feature 99: -0.0019476248314716105
Feature 100: -0.02663732407540605
Feature 101: -0.09151148027595961
Feature 102: 0.22878377493045934
Feature 103: -0.3797191417963548
Feature 104: -0.04837696075018861
Feature 105: -0.236605313839272
Feature 106: -0.16263775321897256
Feature 107: -0.05177246309766145

Feature 108: 0.09271151685381045
Feature 109: -0.08462699015585042
Feature 110: 0.011777409381877081
Feature 111: 0.12665575520930622
Feature 112: 0.06211336273491046
Feature 113: -0.09902485017430385
Feature 114: 0.22790881331916715
Feature 115: -0.01051820788181605
Feature 116: 0.06269517523106258
Feature 117: 0.012042221984463239
Feature 118: -0.020191980371916954
Feature 119: 0.050652356969778364
Feature 120: -0.08682446500970788
Feature 121: -0.0019306211665625936
Feature 122: -0.06782323803593143
Feature 123: 0.10662624878819472
Feature 124: -0.07586128702394304
Feature 125: 0.002956445679834007
Feature 126: -0.06913929371633959
Feature 127: 0.07812213390930312
Feature 128: 0.058890056053435044
Feature 129: 0.04850917342325624
Feature 130: 0.027641938796109603
Feature 131: -0.019533472149149272
Feature 132: 0.09532395135513892
Feature 133: -0.016545366690697547
Feature 134: 0.025445881525372388
Feature 135: -0.02322843100205645
Feature 136: 0.09029293998227748
Feature 137: -0.026893980526577474
Feature 138: -0.04952186178765117
Feature 139: -0.021869866027504667
Feature 140: 0.03317080460097313
Feature 141: 0.1494591105182266
Feature 142: 0.05731203219091363
Feature 143: 0.0356422080114566
Feature 144: -0.07173164063541594
Feature 145: -0.022045882153756403
Feature 146: -0.010664483706202666
Feature 147: -0.02147972624194419
Feature 148: 0.06323076812043163
Feature 149: 0.00957821039601527
Feature 150: -0.00936808914755207
Feature 151: -0.0019031055166528316
Feature 152: -0.013471040483338545
Feature 153: 0.11255441758853961
Feature 154: -0.0005385739154809449
Feature 155: -0.032621434471895056

Feature 156: -0.05374193561302824
Feature 157: -0.026178705072297984
Feature 158: 0.10830254068210612
Feature 159: 0.08569817995959048
Feature 160: -0.016862552457431283
Feature 161: -0.012235358632798905
Feature 162: 0.03883656626044754
Feature 163: 0.09905684451149654
Feature 164: -0.023589540089735225
Feature 165: -0.00796903158631394
Feature 166: -0.03229678115210392
Feature 167: -0.0644168147923317
Feature 168: -0.028768302030496407
Feature 169: -0.007634748697783563
Feature 170: -0.11532994993009733
Feature 171: 0.028110665811681892
Feature 172: 0.2642428627438591
Feature 173: 0.04524496580099281
Feature 174: -0.006266109807467434
Feature 175: -0.008299758528714184
Feature 176: -0.04284825141706437
Feature 177: -0.2522410803607187
Feature 178: -0.012853007260284755
Feature 179: -0.20742016079540251
Feature 180: -0.0004676076467601424
Feature 181: -0.3779598486281197
Feature 182: -0.3116741021396316
Feature 183: 0.3584064647386705
Feature 184: -0.1711342783883356
Feature 185: -0.03255529197627278
Feature 186: 0.3808568536358883
Feature 187: -0.23377124238569413
Feature 188: 0.2115548254284069
Feature 189: -0.10261552773545687
Feature 190: -0.050270633154290104
Feature 191: 0.006245277506623913
Feature 192: 0.013077405024409634
Feature 193: 0.05747379528502867
Feature 194: -0.18989849316692547
Feature 195: 0.2906753806282403
Feature 196: -0.26123638376979824
Feature 197: -0.14658094079696196
Feature 198: 0.017926087302290047
Feature 199: -0.3557252505452521
Feature 200: -0.18418572388036472
Feature 201: -0.052032699754849414
Feature 202: 0.3042548986374411
Feature 203: -0.21825854630561808

Feature 204: -0.15646210107575373
Feature 205: 0.9546234802461391
Feature 206: -0.31288048945644426
Feature 207: -0.10982998258434534
Feature 208: -0.021343768544889303
Feature 209: 0.3930343887344269
Feature 210: 0.44868014727048255
Feature 211: -0.16388720527937944
Feature 212: -0.4752395213146588
Feature 213: -0.25012060471531816
Feature 214: 0.7743021132764532
Feature 215: -0.07483979859517287
Feature 216: -0.004302634471465707
Feature 217: -0.0703098295623435
Feature 218: 0.07364704778974596
Feature 219: -0.4218773713216358
Feature 220: -0.013708788763434065
Feature 221: -0.004478770737315066
Feature 222: -0.12691076838029391
Feature 223: -0.09680012827560244
Feature 224: -0.01977036176441525
Feature 225: -0.01621280709177409
Feature 226: -0.5493437789911658
Feature 227: 0.08704747697219486
Feature 228: 0.7654831253569575
Feature 229: 0.11202465434032964
Feature 230: -0.08277944697318453
Feature 231: -0.052650475403919254
Feature 232: 0.894037641846433
Feature 233: 0.1460620192421906
Feature 234: -0.19800276612836837
Feature 235: -0.6948319080225073
Feature 236: 0.1851754845009024
Feature 237: -0.6304307613379408
Feature 238: 0.13581805329664046
Feature 239: 0.13355474139684084
Feature 240: -0.6583102837188917
Feature 241: -0.21344123035589618
Feature 242: -0.17370634777102206
Feature 243: 0.6141090083815869
Feature 244: 0.18303271613115754
Feature 245: -0.5035873097501229
Feature 246: 0.007720165803887399
Feature 247: -0.010728649024326855
Feature 248: -0.2778435615032377
Feature 249: 0.3618146074216819
Feature 250: 0.022410986679478398
Feature 251: -0.8074257125451327

Feature 252: -0.0249676151634989
Feature 253: 0.5661966831614672
Feature 254: 0.7631677549024414
Feature 255: -0.1903684935206478
Feature 256: -0.12158726501530817
Feature 257: -0.05938152906821181
Feature 258: -0.04388628294569333
Feature 259: 0.4671685101952613
Feature 260: -0.39705877688527474
Feature 261: -0.36098952655809446
Feature 262: -0.27150719374312515
Feature 263: -0.2783567081320189
Feature 264: 0.7947303082932431
Feature 265: -0.00851225710592848
Feature 266: -0.05811737718486722
Feature 267: -0.2054559385136431
Feature 268: 0.07592230893891273
Feature 269: 0.014716372599437706
Feature 270: -0.06110027190699465
Feature 271: -0.04909771178563881
Feature 272: -0.023416313227494417
Feature 273: 0.014928973370460159
Feature 274: -0.06276156771642369
Feature 275: -0.18742967213401537
Feature 276: -0.21006915847551208
Feature 277: -0.530958893999313
Feature 278: -0.974245846692165
Feature 279: -0.12814335894854495
Feature 280: -0.26951386997274895
Feature 281: -0.1031339634264413
Feature 282: 0.12547939834491392
Feature 283: 0.40705798144668565
Feature 284: 0.45277638538694565
Feature 285: 0.7468810119922322
Feature 286: 0.49949490909601985
Feature 287: 0.031252081170436145
Feature 288: -0.0399910933778012
Feature 289: -0.0009028016767908997
Feature 290: 0.27884100876042595
Feature 291: -0.027892559062541605
Feature 292: 0.06514693907048193
Feature 293: -0.016984040917977807
Feature 294: -0.17247774225591156
Feature 295: 0.04595247986403036
Feature 296: 0.03351328778499169
Feature 297: -0.0044581636989632165
Feature 298: 0.03823155544626725
Feature 299: -0.4213196562568938

Feature 300: 0.06657328379023501
Feature 301: -0.24802017505632493
Feature 302: -0.16077333341275155
Feature 303: -0.33454111852617424
Feature 304: -0.13805844983741505
Feature 305: -0.11340440431143847
Feature 306: 0.030292672642858717
Feature 307: 0.10678462525765803
Feature 308: 0.10612091568831522
Feature 309: 0.14112657953446017
Feature 310: 0.18100562169839823
Feature 311: 0.018836558374678982
Feature 312: -0.08646945114303864
Feature 313: 0.038177685326715415
Feature 314: 0.015671526508301035
Feature 315: -0.06834347035384863
Feature 316: -0.04982016623592987
Feature 317: 0.06512599929639756
Feature 318: 0.0619691366359894
Feature 319: -0.09956433817607557
Feature 320: 0.012365417361306115
Feature 321: -0.05801513135863653
Feature 322: -0.031380657501948114
Feature 323: -0.050416113972515955
Feature 324: -0.15563198720473786
Feature 325: -0.09402742692005556
Feature 326: -0.08879373240437304
Feature 327: -0.01267843832045426
Feature 328: -0.02444899847446237
Feature 329: 0.015114671039921866
Feature 330: -0.07512567787140241
Feature 331: 0.10228791251891632
Feature 332: -0.04311721655341702
Feature 333: 0.025903227459928978
Feature 334: -0.06558318500548305
Feature 335: 0.15855221548851772
Feature 336: -0.023979634912457574
Feature 337: -0.139448848349832
Feature 338: -0.049872848628188034
Feature 339: 0.22797794817134379
Feature 340: 0.1694727174144834
Feature 341: 0.06608605592084399
Feature 342: 0.02240133471364671
Feature 343: -0.12228298576871563
Feature 344: -0.004648396672730957
Feature 345: -0.1902956335280348
Feature 346: 0.013140711451130998
Feature 347: 0.20205957840345576

```
    Feature 348: 0.08074242841234537
    Feature 349: 0.07582189885693509
    Feature 350: 0.02069845622210599
    Feature 351: 0.012825139506438522
    Feature 352: -0.03800303488009677
    Feature 353: -0.19617702737915343
    Feature 354: 0.20083909726203564
    Feature 355: 0.4622942710784547
    Feature 356: 0.33843680444369284
    Feature 357: 0.4703095128369804
    Feature 358: 0.1650548830588866
    Feature 359: 0.3214152257001155
    Feature 360: -0.24345473383053445
    Feature 361: -0.27123235710361693
    Feature 362: -0.339540168139215
    Feature 363: -0.4449869432994351
    Feature 364: -0.3878879128268408
    Feature 365: -0.20467439539028917
    Feature 366: -0.13609208172616308
    Feature 367: -0.1167271812730456
    Feature 368: -0.09559594851687718
    Feature 369: -0.058349387532341694
    Feature 370: -0.1359031013231112
    Feature 371: 0.007405198824320366
    Feature 372: -0.06346349197093484
    Feature 373: -0.11032612396779229
    Feature 374: 0.1837331776833145
    Feature 375: 0.01278555419779257
    Feature 376: 0.0573510051061796
    Feature 377: 0.2071622054423348
Class 1 Coefficients:
    Feature 0: -0.05957528170085271
    Feature 1: 0.07674562485345318
    Feature 2: 0.03392740803272069
    Feature 3: -0.03422949697457875
    Feature 4: -0.045210668232899834
    Feature 5: 0.04752067396317347
    Feature 6: -0.019100740474685466
    Feature 7: 0.16612945816693617
    Feature 8: 0.023211867029564383
    Feature 9: -0.09816372535276728
    Feature 10: 0.019731946997970966
    Feature 11: 0.04272545628675701
    Feature 12: 0.026287851390009886
    Feature 13: -0.015159527770342879
    Feature 14: 0.2458991402674525
    Feature 15: -0.1597717720273248
    Feature 16: 0.003658110388623615
```

```
Feature 17: 0.005481447718430512
Feature 18: 0.1843031876491251
Feature 19: 0.040103508585478884
Feature 20: -0.15178438347967127
Feature 21: -0.042475383279589045
Feature 22: 0.4310708747980029
Feature 23: 0.26491893842580194
Feature 24: 0.3910299221065569
Feature 25: 0.4449705891661036
Feature 26: 0.7766183044985158
Feature 27: -0.0503933260358845
Feature 28: -0.07472356177643906
Feature 29: -0.4783573569245558
Feature 30: -1.0866838082251338
Feature 31: -0.01078783514805694
Feature 32: -0.02488946992606695
Feature 33: -0.3466489900686779
Feature 34: -0.21881682216806342
Feature 35: -0.40233878061082473
Feature 36: -0.5624658655103123
Feature 37: 0.3041834334601947
Feature 38: 0.6208049136540262
Feature 39: 0.15905101972701235
Feature 40: 0.01659589178999674
Feature 41: 0.019173202146865034
Feature 42: -0.0767119729879738
Feature 43: 0.022115513794723558
Feature 44: -0.09750060265579817
Feature 45: 0.18204520512264377
Feature 46: -0.018783819597605583
Feature 47: 0.07637555605351147
Feature 48: 0.04228623319527694
Feature 49: 0.0232639412420275
Feature 50: -0.10790698584776917
Feature 51: 0.055590017186298094
Feature 52: 0.17283632660649972
Feature 53: -0.01884228274384214
Feature 54: -0.0557463593770551
Feature 55: -0.01551605503252394
Feature 56: 0.015164646140192107
Feature 57: 0.018138705755089007
Feature 58: -0.005633866011994803
Feature 59: -0.10661615531421886
Feature 60: -0.0409535902503833
Feature 61: 0.03986895079922619
Feature 62: 0.13384185887120628
Feature 63: -0.0014859099297384118
Feature 64: 0.06343236800008177
```

```
Feature 65: 0.04702874476006618
Feature 66: -0.007301988064096831
Feature 67: -0.08328557478634409
Feature 68: -0.049707051397030286
Feature 69: 0.024139421961534117
Feature 70: 0.14372216889772127
Feature 71: 0.1930500741413242
Feature 72: 0.25108584915700766
Feature 73: -0.05490909494602238
Feature 74: -0.26872205011726
Feature 75: 0.05060378023159725
Feature 76: -0.02069095464362418
Feature 77: 0.0973124162385865
Feature 78: 0.055238218328914825
Feature 79: -0.12202973351631138
Feature 80: 0.006366036268647814
Feature 81: -0.05076236170066186
Feature 82: 0.18211109557742933
Feature 83: 0.26895060185201897
Feature 84: 0.08498837976148886
Feature 85: 0.12775311779865625
Feature 86: -0.04656022400137925
Feature 87: 0.059542681533037364
Feature 88: -0.2511090809764718
Feature 89: -0.0178399562025612
Feature 90: 0.5441513691240081
Feature 91: -0.014746742945812432
Feature 92: -0.8006990620782245
Feature 93: 0.048557687223519545
Feature 94: -0.1440939844215483
Feature 95: 0.23207847664022122
Feature 96: 0.08585821616911309
Feature 97: 0.03404474475822022
Feature 98: -0.0970042605948756
Feature 99: 0.03458251904330366
Feature 100: -0.22812004186119808
Feature 101: -0.045647640927146856
Feature 102: 0.04768713118041214
Feature 103: -0.25348818541699464
Feature 104: 0.291032939685551
Feature 105: -0.05854038314795919
Feature 106: 0.37728439154970134
Feature 107: -0.08132135677502858
Feature 108: 0.08939887050053727
Feature 109: 0.19579582081615693
Feature 110: 0.04974215451474927
Feature 111: -0.05360812522603452
Feature 112: -0.15545074406882714
```

```
Feature 113: 0.15314311623528468
Feature 114: -0.24884698699277077
Feature 115: -0.008821870448462197
Feature 116: -0.06843884053333482
Feature 117: -0.13508868414895786
Feature 118: 0.02741417791790606
Feature 119: -0.019606973294377977
Feature 120: 0.029116390852753655
Feature 121: 0.01617002041762131
Feature 122: 0.07600792174847733
Feature 123: -0.07054926617927623
Feature 124: -0.02408883273055666
Feature 125: 0.1365194487528465
Feature 126: -0.022245040269543073
Feature 127: 0.23156707392066056
Feature 128: 0.11604430658657994
Feature 129: -0.03988082413030431
Feature 130: -0.1386032520682975
Feature 131: 0.05805998324049904
Feature 132: -0.02546285996471322
Feature 133: 0.08479759488744344
Feature 134: -0.0018117717450071584
Feature 135: 0.051149703833964555
Feature 136: -0.09178628620505691
Feature 137: 0.016486388641681064
Feature 138: -0.028307979552841665
Feature 139: -0.0618880172100024
Feature 140: -0.018666236407304342
Feature 141: 0.08814363470233803
Feature 142: -0.1303158310903653
Feature 143: 0.04048615403908619
Feature 144: -0.05106820194540437
Feature 145: 0.07514937493353815
Feature 146: -0.10223175259022553
Feature 147: -0.08938749310673973
Feature 148: -0.021338810143205766
Feature 149: 0.0205313957386202
Feature 150: -0.03396905938779611
Feature 151: 0.022111271775901654
Feature 152: -0.05675053028338729
Feature 153: 0.16755263121637887
Feature 154: 0.041909704145651754
Feature 155: -0.019789740210410307
Feature 156: 0.0901973004211117
Feature 157: -0.01683146631653282
Feature 158: 0.1320753862871167
Feature 159: -0.08032181492780367
Feature 160: 0.044898169015917734
```

Feature 161: -0.021927304190023904
Feature 162: -0.07421468090099934
Feature 163: -0.07254628667448249
Feature 164: -0.0037264530623146103
Feature 165: 0.06030610691865171
Feature 166: 0.015738652657752365
Feature 167: 0.07194467707761111
Feature 168: 0.017268900112612732
Feature 169: -0.0017154608202421377
Feature 170: 0.06732124085151257
Feature 171: 0.07152180607604948
Feature 172: 0.09920295287151104
Feature 173: 0.009644205531182457
Feature 174: -0.014815322999762844
Feature 175: 0.22203896489408817
Feature 176: 0.02222072693136846
Feature 177: -0.14500737177378406
Feature 178: -0.00817578060843606
Feature 179: -0.11934751405344929
Feature 180: -0.009059130300956746
Feature 181: -0.2592673472818828
Feature 182: 0.07126971417165309
Feature 183: 0.1636532745484232
Feature 184: 0.3205120571433975
Feature 185: 0.183008155559585
Feature 186: 0.21381840136142197
Feature 187: -0.46848088827756695
Feature 188: 0.0737640630656075
Feature 189: -0.09077108594226516
Feature 190: 0.159687777488469
Feature 191: 0.3890189253561565
Feature 192: 0.07319621681889038
Feature 193: 0.00642296576498841
Feature 194: -0.025870946608978547
Feature 195: -0.232167615031934
Feature 196: 0.12603642701697124
Feature 197: -0.8265552629926126
Feature 198: 0.21789176071216493
Feature 199: 0.5048532418427919
Feature 200: -0.04812223423503645
Feature 201: 0.2039324713149968
Feature 202: 0.10734066451490001
Feature 203: -0.5248865811678742
Feature 204: 0.3308179660968545
Feature 205: -0.09205115030560984
Feature 206: -0.11981994551023234
Feature 207: -0.7487420800551737
Feature 208: 0.15356036358492323

```
Feature 209: 0.20603147801942476
Feature 210: 0.3063882141768763
Feature 211: 0.2692234115634709
Feature 212: 0.31333014335099385
Feature 213: -0.9432055444612848
Feature 214: 0.8485048046033933
Feature 215: 0.4247951300860878
Feature 216: 0.18279374089980185
Feature 217: -0.27215960321760024
Feature 218: 0.08563593963115342
Feature 219: -0.1472864295622513
Feature 220: -0.003756818664763577
Feature 221: -0.04362829596571183
Feature 222: -0.549359454850433
Feature 223: -0.29494941193257496
Feature 224: 0.06507322961184066
Feature 225: 0.061369517954693704
Feature 226: -0.5161215713533075
Feature 227: 0.18896806205683023
Feature 228: 0.8306191161253887
Feature 229: 0.3733548605071051
Feature 230: -0.4859593155658684
Feature 231: 0.06131204831975531
Feature 232: 1.0398616965761616
Feature 233: 0.2553438268205008
Feature 234: 0.3019011905429459
Feature 235: -0.4068258701821715
Feature 236: 0.2808230061283411
Feature 237: -1.41335491411981
Feature 238: 0.32501846634541165
Feature 239: 1.0310820307794357
Feature 240: -1.7518915567180824
Feature 241: -0.3732578229735788
Feature 242: -0.6847168336871658
Feature 243: 0.7233534112467028
Feature 244: 0.0758710283383161
Feature 245: -1.4863443857994414
Feature 246: 0.15931959180830982
Feature 247: -0.039147290769226774
Feature 248: 0.11054520920243949
Feature 249: 0.3952032259233648
Feature 250: 0.15910858070990133
Feature 251: -0.14701142064308864
Feature 252: 0.3842680365162274
Feature 253: 0.5328960368531991
Feature 254: 0.4333340652844322
Feature 255: 0.07622970029817504
Feature 256: -0.2000550378500699
```

Feature 257: -0.18776458227449191
Feature 258: -0.6909098525584308
Feature 259: 0.4490678063178882
Feature 260: 0.0835450585533578
Feature 261: -0.18913906579724613
Feature 262: -0.16478868068848346
Feature 263: -0.07128675212268525
Feature 264: 0.4536009081397564
Feature 265: 0.08883496000950607
Feature 266: 0.08852738560118976
Feature 267: 0.010597705261252758
Feature 268: -0.1562521575479886
Feature 269: 0.10483428611822737
Feature 270: -0.005016176437214295
Feature 271: -0.011572623069952052
Feature 272: 0.018887870185191072
Feature 273: 0.13249444061201585
Feature 274: 0.001748668152155057
Feature 275: -0.19423236240779315
Feature 276: -0.6804830253071603
Feature 277: -0.5182277988314371
Feature 278: -0.3045790709813593
Feature 279: -0.16666379472908469
Feature 280: -0.023827431821745102
Feature 281: 0.11143469244451414
Feature 282: 0.1944287564055996
Feature 283: 0.3226161249245593
Feature 284: 0.321545878744527
Feature 285: 0.5053151079449503
Feature 286: 0.4800459405861003
Feature 287: 0.032323791347450274
Feature 288: 0.066768740479724
Feature 289: -0.009923369356642153
Feature 290: 0.20793324878645214
Feature 291: 0.0512518495758152
Feature 292: -0.16681137296524345
Feature 293: 0.0024197872037527977
Feature 294: 0.04349248311308132
Feature 295: -0.027723788326147344
Feature 296: 0.2184397011153856
Feature 297: -0.017179884376529852
Feature 298: 0.03555662926005173
Feature 299: -0.2108364739444252
Feature 300: 0.21162513864729834
Feature 301: -0.07508295920509904
Feature 302: -0.10862278121568095
Feature 303: 0.013298977834820487
Feature 304: -0.04683831236406831

Feature 305: -0.05446090872805579
Feature 306: 0.045091721581120646
Feature 307: 0.0767915693289544
Feature 308: 0.12304417517543456
Feature 309: 0.08626334969838043
Feature 310: 0.001974388131289046
Feature 311: 0.025218694132136444
Feature 312: -0.016168340688894875
Feature 313: -0.03199104187225785
Feature 314: 0.00399136391131445
Feature 315: 0.031173902007565844
Feature 316: 0.00199345015005439
Feature 317: 0.029914154422087294
Feature 318: 0.05255860610867012
Feature 319: 0.016735033185947327
Feature 320: -0.03301955279149327
Feature 321: -0.008790957968310345
Feature 322: 0.06492686884537277
Feature 323: -0.05686083337499518
Feature 324: -0.06338359473112863
Feature 325: 0.01241376328483788
Feature 326: 0.031525722717592436
Feature 327: 0.002352537595219046
Feature 328: -0.025597927874838187
Feature 329: 0.018035060512782742
Feature 330: -0.1287133067162763
Feature 331: -0.0839218362456307
Feature 332: -0.016120540592983575
Feature 333: 0.02794620312597181
Feature 334: 0.12658282469673296
Feature 335: 0.005315159935883021
Feature 336: 0.009324781445447667
Feature 337: 0.0766783107890723
Feature 338: -0.0036868832952231096
Feature 339: -0.0244295249987545
Feature 340: -0.059613424644648895
Feature 341: -0.013728984701038498
Feature 342: 0.030723497195611894
Feature 343: -0.03931768421623913
Feature 344: -0.009791882523493533
Feature 345: 0.03338363703834135
Feature 346: 0.08184447297881585
Feature 347: -0.0392922128885351
Feature 348: 0.10537991554825671
Feature 349: 0.057012686467883975
Feature 350: -0.02179507811433438
Feature 351: 0.005984773697964092
Feature 352: -0.014631307066831205

```
    Feature 353: 0.11292385439673719
    Feature 354: -0.08501194209027331
    Feature 355: 0.17664204681605758
    Feature 356: 0.15748774425982537
    Feature 357: 0.06209706364980317
    Feature 358: 0.36045648243882555
    Feature 359: 0.12785640698228137
    Feature 360: 0.005141751317656952
    Feature 361: -0.104585343276227
    Feature 362: -0.1321007124980609
    Feature 363: -0.11028947047389956
    Feature 364: -0.15327386441430624
    Feature 365: -0.0927950240643902
    Feature 366: 0.07352223388519724
    Feature 367: 0.032667592383659995
    Feature 368: 0.08734567368456182
    Feature 369: 0.06128995336488106
    Feature 370: -0.028611139349914466
    Feature 371: -0.06171431823330274
    Feature 372: -0.08991050815119797
    Feature 373: -0.054504381676054627
    Feature 374: -0.127452861512172
    Feature 375: -0.03263013320341335
    Feature 376: -0.02428058948206642
    Feature 377: 0.08919551908471324
Class 2 Coefficients:
    Feature 0: -0.0242983427615533
    Feature 1: -0.019913914633941772
    Feature 2: 0.07079438436405491
    Feature 3: 0.04872619752358926
    Feature 4: 0.3026143565556702
    Feature 5: 0.0778706247611449
    Feature 6: 0.05317157445114492
    Feature 7: -0.1606599129989805
    Feature 8: 0.02902904627488914
    Feature 9: -0.03930790091325881
    Feature 10: 0.030078530289436207
    Feature 11: -0.062208642322602085
    Feature 12: 0.026998738626070842
    Feature 13: 0.08172763048641898
    Feature 14: 0.16984760547268096
    Feature 15: -0.019755108921277433
    Feature 16: 0.03196618370865222
    Feature 17: -0.013241113217686567
    Feature 18: 0.08427464897415313
    Feature 19: -0.07820642378625212
    Feature 20: 0.000339298840189958
    Feature 21: 0.06615679780213488
```

```
Feature 22: 0.2127691749115154
Feature 23: 0.17430255314579696
Feature 24: 0.20660332845879611
Feature 25: -0.1910962627150225
Feature 26: 0.2590765612174913
Feature 27: 0.033366045413764
Feature 28: -0.293263284657977
Feature 29: -0.3310110341690892
Feature 30: 0.2597279027994526
Feature 31: 0.014802709650572609
Feature 32: -0.027581245311265802
Feature 33: -0.06135628406611916
Feature 34: 0.259864904289041
Feature 35: 0.16640655972609025
Feature 36: 0.04525564496370012
Feature 37: -0.47297205263911196
Feature 38: 0.22791174063411693
Feature 39: -0.13269870366666023
Feature 40: -0.0363096769212953
Feature 41: -0.023582898446885995
Feature 42: -0.019135925212911783
Feature 43: 0.08531963466388254
Feature 44: 0.19171385995048948
Feature 45: -0.2967082682318578
Feature 46: 0.007831200181826417
Feature 47: 0.18050980687741794
Feature 48: 0.14844668881761108
Feature 49: 0.10844550667397712
Feature 50: 0.03495976562718043
Feature 51: -0.03138143599435041
Feature 52: 0.0731664191803716
Feature 53: 0.07118209764109167
Feature 54: 0.01370262053754167
Feature 55: -0.01634971606053556
Feature 56: 0.09599293659603765
Feature 57: -0.007072528348885716
Feature 58: 0.14683084782571607
Feature 59: -0.12934294890139117
Feature 60: 0.06241374666480958
Feature 61: -0.0718481886626735
Feature 62: 0.11143297151300793
Feature 63: 0.01654998412847982
Feature 64: 0.022198549130058018
Feature 65: 0.03532169686831417
Feature 66: 0.03965957181825108
Feature 67: -0.022994995969142416
Feature 68: -0.11748500641561117
Feature 69: 0.14148897536445434
```

```
Feature 70: 0.2353694830602821
Feature 71: -0.06372372995215615
Feature 72: 0.16851286002442362
Feature 73: -0.017102472120663834
Feature 74: 0.06191308100478053
Feature 75: -0.06454150668360661
Feature 76: -0.03094751574497715
Feature 77: 0.08393497421285519
Feature 78: -0.03730811743285152
Feature 79: 0.032834993324698594
Feature 80: 0.00921435318722086
Feature 81: 0.20732133816536785
Feature 82: 0.0242476757918545
Feature 83: 0.11676315385300415
Feature 84: -0.02002231094487382
Feature 85: -0.220200710348876
Feature 86: 0.051920143472352166
Feature 87: -0.03799220478531693
Feature 88: 0.20371845876184128
Feature 89: 0.05102059263977877
Feature 90: 0.2711371536077328
Feature 91: 0.02303556950253564
Feature 92: -0.11351926356494395
Feature 93: 0.3021055399455035
Feature 94: -0.06956861697905244
Feature 95: 0.11757133501863155
Feature 96: -0.03634145147983872
Feature 97: 0.09291990997825864
Feature 98: -0.211520631122531
Feature 99: 0.009563967667596329
Feature 100: 0.14348141885470783
Feature 101: 0.14706069459377888
Feature 102: 0.09082556728198775
Feature 103: 0.049699726122689844
Feature 104: -0.04742051922837493
Feature 105: 0.1494396141578167
Feature 106: 0.030271418843662818
Feature 107: -0.07181623132823085
Feature 108: 0.159089088433445
Feature 109: -0.09068348797606247
Feature 110: -0.1239619311593896
Feature 111: -0.019303725053376342
Feature 112: 0.047364076189747945
Feature 113: 0.013066046849689612
Feature 114: 0.018374706359505222
Feature 115: -0.01672341721856471
Feature 116: 0.07069249044328695
Feature 117: 0.1252374610880912
```

Feature 118: -0.037705740256997466
Feature 119: -0.012383578040034982
Feature 120: 0.08424450851457135
Feature 121: -0.015881346200136925
Feature 122: 0.09020269084884952
Feature 123: 0.06553282948076142
Feature 124: 0.006411362493153305
Feature 125: 0.05837632346916169
Feature 126: 0.014120653181897633
Feature 127: 0.06858600181217593
Feature 128: 0.11736729820002614
Feature 129: -0.03439411031285441
Feature 130: 0.058588813243616755
Feature 131: -0.051808148479212446
Feature 132: -0.017342592737856315
Feature 133: -0.2117337533932902
Feature 134: -0.04577852997645509
Feature 135: 0.006152164924819113
Feature 136: -0.033055954212266855
Feature 137: -0.010589658550323556
Feature 138: 0.04786058408584766
Feature 139: -0.17265071014372416
Feature 140: -0.21828441794838435
Feature 141: 0.01824425336614282
Feature 142: 0.1376937581505962
Feature 143: -0.04271761575000146
Feature 144: 0.1150468527039462
Feature 145: 0.0321188907681447
Feature 146: -0.018941626286660462
Feature 147: 0.09879256616393299
Feature 148: 0.10363127924287718
Feature 149: 0.03405344772546442
Feature 150: 0.07141344771681948
Feature 151: -0.010841343643588918
Feature 152: 0.011876458056540473
Feature 153: -0.14084403398614787
Feature 154: -0.02787333942159173
Feature 155: 0.03723156253976386
Feature 156: -0.026961225670159333
Feature 157: 0.06775689962008247
Feature 158: -0.19478548640339824
Feature 159: -0.06999672463285705
Feature 160: 0.009622553718883788
Feature 161: 0.019359649893628133
Feature 162: -0.08131219503330772
Feature 163: 0.009058157108930553
Feature 164: 0.07621723296166014
Feature 165: -0.08324925210549673

Feature 166: 0.06788375382698432
Feature 167: 0.15123746885292325
Feature 168: 0.06151081068982527
Feature 169: -0.027631905055639137
Feature 170: 0.10069998737322125
Feature 171: -0.08337244728813535
Feature 172: -0.05311598351994195
Feature 173: -0.028392806798232757
Feature 174: 0.04494146460155175
Feature 175: -0.1406040948393773
Feature 176: 0.12005683486539076
Feature 177: 0.1355857595837877
Feature 178: 0.054202811786961215
Feature 179: -0.09913312274740589
Feature 180: 0.026750338437461847
Feature 181: -0.06014600290464458
Feature 182: 0.040219292255267124
Feature 183: 0.009866751973814309
Feature 184: -0.0591562517820496
Feature 185: 0.04368851777961342
Feature 186: 0.13506599176429548
Feature 187: -0.33816327147476766
Feature 188: -0.08394659053876784
Feature 189: -0.02961003943764622
Feature 190: 0.10309731968257176
Feature 191: 0.22245979267187832
Feature 192: 0.079301395337343
Feature 193: 0.14792858572953022
Feature 194: 0.21334230326243173
Feature 195: -0.055766247726318476
Feature 196: 0.049935396443003814
Feature 197: -0.12815468795511756
Feature 198: 0.07723274826956275
Feature 199: 0.16652631363486708
Feature 200: 0.48834788914142324
Feature 201: 0.14210776119313456
Feature 202: 0.08526838898451637
Feature 203: -0.3311482095756732
Feature 204: 0.09365303943978336
Feature 205: -0.5415157616836048
Feature 206: 0.21190180517915191
Feature 207: 0.22364237299175715
Feature 208: 0.04302719143633433
Feature 209: 0.15704755389559483
Feature 210: -0.1641232874956038
Feature 211: 0.06882233128046986
Feature 212: -0.09667923634458604
Feature 213: -0.2740026423620436

Feature 214: 0.17005505713692584
Feature 215: -0.20780675080676486
Feature 216: -0.2841022805408245
Feature 217: 0.015635419073016948
Feature 218: 0.052814399780499593
Feature 219: -0.11084826200665784
Feature 220: -0.03710882053873801
Feature 221: -0.02774432614101237
Feature 222: -0.33061105928983014
Feature 223: -0.058145581263235005
Feature 224: -0.0025995838260463805
Feature 225: -0.06815584220793623
Feature 226: -0.14290019625484238
Feature 227: -0.1203262568077591
Feature 228: -0.20984160825045073
Feature 229: -0.17818480467195802
Feature 230: -0.10665221067266145
Feature 231: -0.13923307155949935
Feature 232: 0.1274555099887243
Feature 233: 0.09888887847234033
Feature 234: 0.1404067968506907
Feature 235: 0.5268636923439914
Feature 236: -0.0937318222084448
Feature 237: 0.11077624454064734
Feature 238: -0.10219871208759757
Feature 239: 0.20483838893749393
Feature 240: 0.254197879207022
Feature 241: -0.04773514992047785
Feature 242: 0.674222145587985
Feature 243: 0.24188018512049456
Feature 244: -0.080769044286982
Feature 245: 0.13092447135341753
Feature 246: 0.2270829356164866
Feature 247: 0.05462394773997248
Feature 248: 0.27263563490341025
Feature 249: -0.3076878864356288
Feature 250: 0.027388727883770267
Feature 251: 0.3175737267325748
Feature 252: 0.21054300694929565
Feature 253: 0.05627123414017946
Feature 254: -0.46622874290837424
Feature 255: -0.1672720642888911
Feature 256: 0.08355968915663904
Feature 257: -0.16588830230341683
Feature 258: -0.014301645749113485
Feature 259: 0.08392812609198878
Feature 260: 0.18215609666384822
Feature 261: 0.01222462106193356

Feature 262: -0.12713793899773504
Feature 263: 0.1852293042930683
Feature 264: -0.16474763963136232
Feature 265: 0.04288956485465723
Feature 266: 0.11850034234692311
Feature 267: 0.131601484921907
Feature 268: -0.10493314703522678
Feature 269: 0.16205001289682341
Feature 270: 0.12535669269293837
Feature 271: 0.08475760538154727
Feature 272: 0.06943715264337438
Feature 273: 0.0038446155002379648
Feature 274: 0.06671219176698916
Feature 275: -0.08975165059470208
Feature 276: 0.03272487745372657
Feature 277: -0.01417568204738572
Feature 278: 0.025838668645961137
Feature 279: -0.04111351269350113
Feature 280: 0.13808902787626087
Feature 281: 0.1310097137269627
Feature 282: 0.1180666139419047
Feature 283: 0.12024245455100722
Feature 284: 0.09664978718274704
Feature 285: -0.018967876386318493
Feature 286: -0.12160561976451312
Feature 287: 0.025284691125555425
Feature 288: -0.02450534406456323
Feature 289: -0.02767789096805845
Feature 290: -0.09808962415554766
Feature 291: -0.02403490168147065
Feature 292: 0.07262321941844282
Feature 293: 0.0015447610025691651
Feature 294: 0.2901205822384718
Feature 295: -0.018136972385844913
Feature 296: -0.09154409898323547
Feature 297: 0.03922932751084939
Feature 298: -0.03658020086477505
Feature 299: 0.21497616588562912
Feature 300: 0.21326052154794975
Feature 301: 0.1368477364371374
Feature 302: 0.011469184920152621
Feature 303: 0.06050404195180343
Feature 304: 0.11595364450748741
Feature 305: 0.009405381462150149
Feature 306: 0.07125235010542273
Feature 307: 0.004004077263726792
Feature 308: 0.05739768077565923
Feature 309: 0.0156139793969122

Feature 310: 0.004507917601772441
Feature 311: 0.02363278262149324
Feature 312: 0.06855272370698692
Feature 313: -0.007050229334194956
Feature 314: 0.028223036304452304
Feature 315: 0.009003087519198087
Feature 316: 0.024932963271848078
Feature 317: 0.06364206138536156
Feature 318: 0.057446418052796995
Feature 319: -0.005646791738123626
Feature 320: 0.004428846901420732
Feature 321: -0.017760840275058375
Feature 322: 0.10070419956890689
Feature 323: 0.06399583302388565
Feature 324: -0.04394541101672747
Feature 325: 0.01870830452482232
Feature 326: 0.07336199804394469
Feature 327: 0.016540025476396485
Feature 328: -0.07927821623340767
Feature 329: 0.05860383815610372
Feature 330: -0.06960819331012237
Feature 331: -0.06981132368534534
Feature 332: 0.0452381554835893
Feature 333: 0.04291458570180475
Feature 334: 0.047205259469974746
Feature 335: 0.09677028756108341
Feature 336: 0.07651724704732404
Feature 337: 0.00442184050990615
Feature 338: 0.014988586959157097
Feature 339: -0.012574683277008458
Feature 340: -0.034008506069158194
Feature 341: -0.008531997486420907
Feature 342: 0.04726039247951493
Feature 343: 0.009373842788681619
Feature 344: -0.03158392153815662
Feature 345: -0.06826236170609828
Feature 346: 0.10735008517658695
Feature 347: 0.006295190368745821
Feature 348: 0.036015907033275014
Feature 349: -0.03356334425651852
Feature 350: -0.07651273487177464
Feature 351: 0.030330494269112915
Feature 352: 0.0026144935465027847
Feature 353: 0.07928258381541738
Feature 354: -0.027613216979653882
Feature 355: -0.1678670607132621
Feature 356: 0.11951135768790416
Feature 357: -0.057200831240137834

```
    Feature 358: 0.07314095153326895
    Feature 359: -0.06605300195843378
    Feature 360: 0.09975271106642948
    Feature 361: 0.05708705425403862
    Feature 362: 0.045714426951787825
    Feature 363: 0.1169105481572371
    Feature 364: -0.049787845251130364
    Feature 365: 0.06966542803990246
    Feature 366: 0.11436387408302547
    Feature 367: 0.06356281714309367
    Feature 368: 0.130065919723998
    Feature 369: 0.10363711745673682
    Feature 370: -0.046816827837441496
    Feature 371: 0.021252407816043286
    Feature 372: -0.069780392933074
    Feature 373: -0.13250204288060782
    Feature 374: -0.08871804751687168
    Feature 375: -0.004438131610462485
    Feature 376: 0.025501405439376464
    Feature 377: 0.02071963755332182
Class 3 Coefficients:
    Feature 0: -0.04540653533378475
    Feature 1: -0.07917934889670925
    Feature 2: 0.01872795479746873
    Feature 3: 0.379894451216528
    Feature 4: -0.44434599569260275
    Feature 5: -0.16130102622827466
    Feature 6: 0.24059810229820547
    Feature 7: 0.10790547457558201
    Feature 8: 0.10763054540966017
    Feature 9: -0.023572097457622447
    Feature 10: -0.26981095352632706
    Feature 11: -0.022604567217277948
    Feature 12: 0.029698867348463297
    Feature 13: 0.028777673960594895
    Feature 14: 0.055912217515070764
    Feature 15: 0.1610565883475839
    Feature 16: 0.0400968012025669
    Feature 17: 0.02726419875514775
    Feature 18: -0.4763395838239103
    Feature 19: -0.04356258537092196
    Feature 20: 0.17991563634558597
    Feature 21: -0.0696979261048335
    Feature 22: -0.3099814559920427
    Feature 23: -0.6691466630803669
    Feature 24: -0.23602165956853757
    Feature 25: -0.7923212323912737
    Feature 26: -1.1757298410519827
```

```
Feature 27: -0.08406931455832116
Feature 28: 0.3220965744382397
Feature 29: 0.961220982471582
Feature 30: -0.04259708775039536
Feature 31: 0.3132894361905359
Feature 32: -0.2360901192818271
Feature 33: 0.23400507179413563
Feature 34: 0.16966446426617232
Feature 35: 0.3425925499127316
Feature 36: 0.7711564102341402
Feature 37: 0.7821610203508718
Feature 38: -0.6326970023989362
Feature 39: 0.21558975459057553
Feature 40: -0.0023811121774886443
Feature 41: -0.025548474537815363
Feature 42: 0.06123022578533352
Feature 43: 0.00014841280726327533
Feature 44: -0.13344567708149582
Feature 45: 0.14347367026790728
Feature 46: -0.13922374462586412
Feature 47: -0.07153586982114311
Feature 48: -0.11887564218798702
Feature 49: 0.044489452694176645
Feature 50: 0.07878367920724588
Feature 51: 0.09600696784520833
Feature 52: -0.15934366669488587
Feature 53: -0.008309285913467345
Feature 54: -0.05272041513901317
Feature 55: -0.012996473806286421
Feature 56: -0.023141992499253847
Feature 57: -0.0011661648297416566
Feature 58: -0.03757523743224339
Feature 59: 0.2353784042408225
Feature 60: 0.020553412288505
Feature 61: 0.036119696774224636
Feature 62: -0.06367638881332312
Feature 63: -0.11255365959205872
Feature 64: 0.15259912508133044
Feature 65: 0.08659544834510018
Feature 66: 0.012660267825311341
Feature 67: 0.049341675105934825
Feature 68: -0.042419281817911894
Feature 69: -0.11562960558015563
Feature 70: -0.09747208119221044
Feature 71: -0.2850774591737237
Feature 72: -0.26137752489248334
Feature 73: 0.3343573912961505
Feature 74: 0.18082050341340378
```

```
Feature 75: -0.00541557733773926
Feature 76: 0.034906630892348155
Feature 77: -0.10841796293280764
Feature 78: 0.007728853524741852
Feature 79: -0.05678803351648498
Feature 80: -0.0910987590390091
Feature 81: 0.18348382594094437
Feature 82: -0.24875285670951153
Feature 83: -0.4563038506725598
Feature 84: -0.018978777194873935
Feature 85: -0.038647134023759924
Feature 86: -0.05471741427698592
Feature 87: -0.10302529824327236
Feature 88: 0.012892538584518384
Feature 89: -0.05189716632330961
Feature 90: 0.7486468958555544
Feature 91: -0.005426489470839446
Feature 92: 0.14933144065037332
Feature 93: -0.1809244465360214
Feature 94: 0.2745772381676171
Feature 95: -0.16053090345625665
Feature 96: -0.019798121189174497
Feature 97: 0.07818472170939783
Feature 98: -0.018126534880941778
Feature 99: -0.02068049185796772
Feature 100: -0.06417224516678713
Feature 101: 0.03360870786562794
Feature 102: -0.47019538152118157
Feature 103: 0.32847363413936587
Feature 104: 0.059503093266325774
Feature 105: 0.06445668092293691
Feature 106: 0.01575683037028438
Feature 107: 0.30664298758924546
Feature 108: -0.09073627704990588
Feature 109: -0.010868006643650762
Feature 110: -0.03352898769716859
Feature 111: 0.007732471031810293
Feature 112: -0.21632781232867074
Feature 113: -0.04623759442174556
Feature 114: 0.029434214037537185
Feature 115: -0.0026619880927367585
Feature 116: -0.011272184008095654
Feature 117: 0.0060192248982243336
Feature 118: 0.0020715704992931307
Feature 119: -0.0027262404879853097
Feature 120: 0.021403305714499353
Feature 121: -0.009376507621569968
Feature 122: -0.08562167502083853
```

```
Feature 123: -0.039582084169187286
Feature 124: -0.054432416590652316
Feature 125: 0.13945630124884761
Feature 126: -0.05050558646241273
Feature 127: -0.12687025913858588
Feature 128: -0.06994537012046509
Feature 129: -0.0015231599958375939
Feature 130: -0.018203935362422263
Feature 131: -0.014700268266215262
Feature 132: -0.008139803541252957
Feature 133: 0.01236055427530814
Feature 134: -0.012695755878836814
Feature 135: -0.003272300669157326
Feature 136: 0.046872862684409766
Feature 137: -0.01682861869470456
Feature 138: 0.08015359967355162
Feature 139: 0.11376060403848756
Feature 140: 0.13578469690922432
Feature 141: 0.03232514800606347
Feature 142: 0.021571102298614954
Feature 143: -0.006972000639524528
Feature 144: 0.14837994587455536
Feature 145: -0.006506914991379419
Feature 146: -0.005870504171693866
Feature 147: -0.02877081774550614
Feature 148: -0.006483463147120155
Feature 149: 0.044082496542016354
Feature 150: -0.005463669697159519
Feature 151: -0.00022578168326521254
Feature 152: 0.08769271730488652
Feature 153: -0.12741900113764248
Feature 154: -0.0010697356134515312
Feature 155: 0.016530839864573356
Feature 156: 0.013210359042476779
Feature 157: 0.1411410068406813
Feature 158: -0.11868924983894004
Feature 159: -0.006679598810949416
Feature 160: -0.01515603508667009
Feature 161: -0.0074887839158903876
Feature 162: 0.21692774380314467
Feature 163: -0.00669381975599593
Feature 164: -0.012967636580738377
Feature 165: -0.009910815486285772
Feature 166: 0.039956959867503916
Feature 167: -0.010890104330538561
Feature 168: -0.10018378783805496
Feature 169: -0.0020184040559256006
Feature 170: -0.08833318597816775
```

Feature 171: 0.008890654771603399
Feature 172: -0.09140604822141193
Feature 173: -0.007799186913294615
Feature 174: -0.001701063086869927
Feature 175: -0.07107974723982861
Feature 176: -0.018680042042724998
Feature 177: 0.14859591731344438
Feature 178: -0.005252929425632033
Feature 179: -0.09791663023964352
Feature 180: -0.0005757028339686479
Feature 181: -0.15350290866683688
Feature 182: -0.20200732717211797
Feature 183: -0.3327227646747982
Feature 184: -0.144125293169803
Feature 185: -0.23523446718214264
Feature 186: -0.4931704551899107
Feature 187: 1.6088256323999057
Feature 188: -0.14932273789153633
Feature 189: 0.0065404457964028
Feature 190: -0.2288546222825929
Feature 191: -0.10065534462619319
Feature 192: -0.050129477285176756
Feature 193: -0.4319647758521669
Feature 194: 0.26852225995789347
Feature 195: -0.04166201247574655
Feature 196: -0.15452986672288133
Feature 197: 0.6136126945461957
Feature 198: -0.24979895064395208
Feature 199: -0.22507934882568295
Feature 200: -0.3012315436540305
Feature 201: -0.09438259137388272
Feature 202: -0.5425833028492474
Feature 203: 1.6091856304188392
Feature 204: -0.16015431325521967
Feature 205: -0.12515877822879662
Feature 206: -0.3112977960310815
Feature 207: 0.17112414816642585
Feature 208: 0.005827687768278137
Feature 209: -0.5530317305809238
Feature 210: 0.05589250296287886
Feature 211: -0.027765273274783897
Feature 212: 0.03552158363122988
Feature 213: 0.6460539639452836
Feature 214: -0.91909992133354
Feature 215: -0.07200177784122656
Feature 216: -0.05986364823162094
Feature 217: -0.005945375676162753
Feature 218: -0.05529678741991005

Feature 219: 0.5187569759568035
Feature 220: -0.04389886854607964
Feature 221: 0.01286776332121043
Feature 222: 0.6722441786610686
Feature 223: 0.03128567255814899
Feature 224: -0.010992680850071565
Feature 225: 0.021437194335176327
Feature 226: 0.7746101275145321
Feature 227: -0.036460838258451644
Feature 228: -0.6449892360969823
Feature 229: 0.09306360103058882
Feature 230: 0.2552941542067937
Feature 231: 0.31802033275249464
Feature 232: -0.827572666421212
Feature 233: -0.45577546404446073
Feature 234: -0.29671341264284623
Feature 235: -0.08017475037882796
Feature 236: 0.06490631407873725
Feature 237: 0.8992989205216433
Feature 238: -0.09661511338680247
Feature 239: -0.46448755659221785
Feature 240: 1.1315705624739523
Feature 241: 0.38264404308056993
Feature 242: -0.2055235660140593
Feature 243: -0.9674662885818988
Feature 244: -0.0710508999701597
Feature 245: 0.8373475343217964
Feature 246: -0.5748866675740699
Feature 247: -0.015269146812205776
Feature 248: -0.09382522040574029
Feature 249: -0.1731467529678871
Feature 250: 0.2263426792811854
Feature 251: 0.6908265436979311
Feature 252: -0.35642854444791455
Feature 253: -0.6048132192949911
Feature 254: -0.2811279929512673
Feature 255: 0.004707577354968875
Feature 256: -0.053540052105342864
Feature 257: 0.025580031861305752
Feature 258: 0.4372249244844343
Feature 259: -0.26866553084362194
Feature 260: 0.04193165824070946
Feature 261: 0.3794419821722798
Feature 262: 0.2909141377952492
Feature 263: 0.03944207839575245
Feature 264: -0.5478832320570675
Feature 265: 0.009532079025119078
Feature 266: 0.05854193056191877

Feature 267: 0.005969463688140901
Feature 268: -0.05782887715049654
Feature 269: -0.08309270794934928
Feature 270: -0.03499185754268964
Feature 271: -0.025666136716201823
Feature 272: 0.03552496839873199
Feature 273: -0.11792339628177481
Feature 274: 0.0761783103172726
Feature 275: 0.22980892243142828
Feature 276: 0.4405285632173199
Feature 277: 0.4321793849587084
Feature 278: 0.6022614292078281
Feature 279: 0.28107660737971296
Feature 280: 0.314503667493674
Feature 281: -0.12045995743475706
Feature 282: -0.20896410396574408
Feature 283: -0.5296440207205011
Feature 284: -0.4166548988624315
Feature 285: -0.7570521282465902
Feature 286: -0.27800353575608866
Feature 287: -0.035298964853936673
Feature 288: -0.02061705487439034
Feature 289: -0.0005420217988930087
Feature 290: -0.049079493449474196
Feature 291: -0.013619776887785455
Feature 292: -0.012177472326530235
Feature 293: 0.044595477506290185
Feature 294: 0.15127072268137517
Feature 295: 0.02670393353866975
Feature 296: -0.07683781626472244
Feature 297: -0.005150714223037841
Feature 298: -0.003920009475237217
Feature 299: -0.12866296292420756
Feature 300: -0.12184017552272118
Feature 301: 0.05496206369806014
Feature 302: 0.020711050152858727
Feature 303: 0.01388183215846576
Feature 304: 0.07216455175490524
Feature 305: -0.036158985330423915
Feature 306: 0.07534509253149266
Feature 307: -0.1418645456713407
Feature 308: -0.0804206327429411
Feature 309: -0.08521489072302534
Feature 310: 0.09467841604534956
Feature 311: 0.02953904861975482
Feature 312: -0.07453296648860414
Feature 313: 0.04066343865675145
Feature 314: -0.02159493054334

Feature 315: 0.15910973450078666
Feature 316: 0.024040826386032485
Feature 317: -0.09341516917713671
Feature 318: -0.04539599459658548
Feature 319: 0.03548814348201885
Feature 320: -0.14551229770172733
Feature 321: 0.08145562418833832
Feature 322: -0.056723569150945535
Feature 323: -0.054510107055738
Feature 324: 0.13158729723468246
Feature 325: 0.05099249443742374
Feature 326: -0.017355026469832932
Feature 327: 0.15649126007653213
Feature 328: 0.1457147426519621
Feature 329: 0.05921235442317841
Feature 330: 0.18418463223249731
Feature 331: 0.0346906722510791
Feature 332: -0.14092666260553188
Feature 333: -0.22835147869648864
Feature 334: -0.04926827865294899
Feature 335: -0.12779372081744736
Feature 336: -0.07703848273459787
Feature 337: 0.23154394362021635
Feature 338: -0.0008061964709731568
Feature 339: -0.10746663885228963
Feature 340: -0.06869811057416648
Feature 341: -0.07279720483362875
Feature 342: -0.03305747539718473
Feature 343: 0.18689789971929724
Feature 344: 0.0611441474059973
Feature 345: 0.10789450012616227
Feature 346: -0.14844364486669387
Feature 347: 0.25630567505519686
Feature 348: -0.1671349668037271
Feature 349: -0.19968553976571984
Feature 350: 0.10786735818022922
Feature 351: -0.3523122124205669
Feature 352: 0.05185266155941716
Feature 353: 0.012387996218997761
Feature 354: -0.07462743122835021
Feature 355: -0.1142864413660483
Feature 356: -0.254182064979931
Feature 357: -0.1842813930162325
Feature 358: -0.11751618240411166
Feature 359: -0.08878120340999454
Feature 360: -0.012309359654950029
Feature 361: 0.12651694253426635
Feature 362: 0.07174951558673924

```
    Feature 363: 0.12538444881613822
    Feature 364: 0.22256799606120464
    Feature 365: 0.17792499753855254
    Feature 366: 0.17619872081338941
    Feature 367: 0.007359453460134123
    Feature 368: 0.052727366921497446
    Feature 369: 0.08629447930795926
    Feature 370: -0.07069039096717127
    Feature 371: 0.1455496269734239
    Feature 372: 0.07973876413191693
    Feature 373: 0.009704124930251128
    Feature 374: 0.010623468366110576
    Feature 375: -0.13583761847194892
    Feature 376: -0.1947137967306691
    Feature 377: -0.11199213503683367
Class 4 Coefficients:
    Feature 0: 0.15625809932726928
    Feature 1: 0.14941709103278983
    Feature 2: -0.1273328174078165
    Feature 3: -0.009223873122239505
    Feature 4: 0.0018061773124233892
    Feature 5: -0.14028950477477933
    Feature 6: -0.04126528039134178
    Feature 7: -0.017972828146206005
    Feature 8: -0.011998232440692912
    Feature 9: -0.0006709626806077963
    Feature 10: -0.10660228748063125
    Feature 11: -0.0001500229327266856
    Feature 12: -0.2132905234243373
    Feature 13: -0.019905232152291893
    Feature 14: -0.010320059125739021
    Feature 15: -0.05446625023640357
    Feature 16: -0.16386293753318307
    Feature 17: -0.006251729684206185
    Feature 18: -0.03324209921305546
    Feature 19: -0.0009342642476727097
    Feature 20: -0.024983284711292967
    Feature 21: 0.0735567482635634
    Feature 22: -0.008592469989684996
    Feature 23: -0.03087844153196431
    Feature 24: -0.01823428504363346
    Feature 25: -0.04192960413861791
    Feature 26: -0.041030585776605386
    Feature 27: -0.004486900229407639
    Feature 28: -0.036324943668818103
    Feature 29: 0.06847392606044944
    Feature 30: -0.01235957158722012
    Feature 31: -0.10618710590004445
```

Feature 32: -0.008984762518235632
Feature 33: -0.010832856268961499
Feature 34: -0.0011226597971393856
Feature 35: -0.05636750402403408
Feature 36: -0.07694341842349466
Feature 37: 0.06045870706506348
Feature 38: -0.12118501243285552
Feature 39: -0.007172143859408468
Feature 40: -5.548648371316978e-05
Feature 41: -0.0018754609735458583
Feature 42: -0.208295303956759
Feature 43: -0.151878390944949
Feature 44: -0.012284438752475589
Feature 45: -0.004872966387776098
Feature 46: -0.17838740033583717
Feature 47: 0.05117210734613682
Feature 48: -0.003085465903361384
Feature 49: 0.028826227355002263
Feature 50: 0.03420961504974288
Feature 51: -0.007172668077083436
Feature 52: -0.1778481673909495
Feature 53: -0.015518130915312641
Feature 54: -0.0006265575733945215
Feature 55: -0.0005164872423635221
Feature 56: -0.0018742440376435863
Feature 57: -4.415190414205858e-05
Feature 58: -0.004010769578915928
Feature 59: 0.03607075667824753
Feature 60: -0.021666501790861464
Feature 61: -0.0010028052628703568
Feature 62: -0.2666625730464134
Feature 63: 0.02411641519567877
Feature 64: -0.2953796195472214
Feature 65: -0.019329768440348786
Feature 66: -0.050660632827226874
Feature 67: -0.14662193202884902
Feature 68: -0.0014821654764194119
Feature 69: -0.004626986396662239
Feature 70: 0.040285057456439566
Feature 71: -0.031022333290726845
Feature 72: -0.22321524923565061
Feature 73: -0.03546203977703833
Feature 74: -0.004943305051977505
Feature 75: -0.0009108789950175516
Feature 76: -0.00033328243985816797
Feature 77: -0.1213324040773471
Feature 78: 0.005293424132131513
Feature 79: -0.008891656794076046

Feature 80: -0.0028958128778148046
Feature 81: -0.029986093657237815
Feature 82: -0.02990797489680159
Feature 83: -0.01908638157723015
Feature 84: -0.0006614593132359992
Feature 85: -0.0023153550610399373
Feature 86: -0.0023862670434265892
Feature 87: -0.005407132290236767
Feature 88: -0.0013626277640893373
Feature 89: -0.002373129303485798
Feature 90: -0.17196223264747895
Feature 91: -0.0006125327195922726
Feature 92: -0.2176245394479956
Feature 93: -0.29545971951883293
Feature 94: 0.0840497033897519
Feature 95: -0.24228961593554266
Feature 96: -0.0016607830250224029
Feature 97: -0.002658923762325061
Feature 98: -0.005583506339251285
Feature 99: -0.0007458793015173339
Feature 100: -0.012468243515243001
Feature 101: -0.002431340501265756
Feature 102: -0.05305403333137904
Feature 103: -0.06574072783172807
Feature 104: -0.005120469878075183
Feature 105: -0.006947651500177416
Feature 106: -0.0043809479668135
Feature 107: 0.009885864208521973
Feature 108: -0.0050350559981347945
Feature 109: -0.00033952133401571886
Feature 110: -0.0036471000847468106
Feature 111: -0.001016144482703383
Feature 112: -0.2905543419638724
Feature 113: -0.0010209262744981886
Feature 114: -0.0011798991823703784
Feature 115: -5.869015910133298e-05
Feature 116: -0.000445910057849407
Feature 117: -0.0012682666355410955
Feature 118: -0.0002920733570324013
Feature 119: -5.0924882616655925e-05
Feature 120: -0.0012205126949926988
Feature 121: -5.801451093817859e-05
Feature 122: -0.006023255618396347
Feature 123: -0.0011220683808779003
Feature 124: -0.001274886837003524
Feature 125: -0.07655244846812535
Feature 126: -0.0026538929348191493
Feature 127: -0.012510338085131398

```
Feature 128: -0.00392249456716937
Feature 129: -6.633276185207962e-05
Feature 130: -0.0004206138219942976
Feature 131: -0.0004508653468235479
Feature 132: -0.00024153219591865253
Feature 133: -0.0016614488570353167
Feature 134: -0.0013989391624021148
Feature 135: -0.00022656582133869402
Feature 136: -0.00033955101631384654
Feature 137: -0.0005589300174490126
Feature 138: -0.0011363734115444701
Feature 139: 0.045650693710571684
Feature 140: -0.001390292058781105
Feature 141: -0.2795709041642845
Feature 142: -0.0015957792675660098
Feature 143: -0.0004273968172768483
Feature 144: -0.001665603857218378
Feature 145: -0.00029605565335602644
Feature 146: -0.0006964127623359231
Feature 147: -0.0005445144086906243
Feature 148: -0.007537447199575714
Feature 149: -0.00041827435145327794
Feature 150: -0.00011284213304143295
Feature 151: -2.2926968384561356e-05
Feature 152: -0.00013932322554351608
Feature 153: -0.005415622123349621
Feature 154: -4.1555762540335506e-05
Feature 155: -0.0007068503929097476
Feature 156: -0.0005584884749925528
Feature 157: -0.007638883998035929
Feature 158: 0.0503431474836038
Feature 159: -0.000329056494137483
Feature 160: -0.0003881791061295859
Feature 161: -0.00040804886641643246
Feature 162: -0.019421443543250892
Feature 163: -0.0001258666668352434
Feature 164: -0.00016130722024804275
Feature 165: -0.00021763517145259782
Feature 166: -0.001638657561551231
Feature 167: -0.0013666884442593403
Feature 168: -0.005766963395597029
Feature 169: -6.524244001282415e-05
Feature 170: -0.004227681341466708
Feature 171: -0.0006193237795422337
Feature 172: -0.02937236115377737
Feature 173: -0.0001368138150184084
Feature 174: -0.00013360744450665931
Feature 175: -0.002017530620855404
```

Feature 176: -0.0003727238203958913
Feature 177: -0.008117738462834425
Feature 178: -0.0001891756990766405
Feature 179: 0.019705498993407237
Feature 180: -3.313506897557838e-05
Feature 181: -0.07155568891425519
Feature 182: 0.1076790042856098
Feature 183: -0.11606604819654005
Feature 184: -0.04650352906569134
Feature 185: -0.05277695243581484
Feature 186: 0.004395008915674643
Feature 187: 0.04639584287545892
Feature 188: 0.12343760509903415
Feature 189: 0.04900786836854058
Feature 190: -0.0475968672314119
Feature 191: -0.1291613584726828
Feature 192: -0.08408994587826639
Feature 193: -0.04707609748054505
Feature 194: -0.33076212347032474
Feature 195: -0.04637193706231229
Feature 196: 0.043817412471306265
Feature 197: 0.07197248521335305
Feature 198: 0.10487769876176158
Feature 199: -0.1110672641122952
Feature 200: -0.04589272196465148
Feature 201: -0.05357120036911465
Feature 202: 0.0022537618991201264
Feature 203: 0.04696377601888766
Feature 204: 0.12958264737463066
Feature 205: 0.04553563158224537
Feature 206: -0.04645255885332439
Feature 207: -0.1255912031342218
Feature 208: -0.08396526128525285
Feature 209: -0.05208236397342807
Feature 210: -0.3827019380660257
Feature 211: -0.04914819961021218
Feature 212: 0.09533467747844027
Feature 213: 0.07222488618882658
Feature 214: -0.19179527272495375
Feature 215: -0.0019693908551235238
Feature 216: -0.0065078969805264005
Feature 217: -0.0020768597726407005
Feature 218: -0.000727409571388026
Feature 219: -0.030645480595440665
Feature 220: -0.0005328474607116507
Feature 221: -0.00021757282559526183
Feature 222: -0.00554844231694218
Feature 223: -0.0038060876088483117

Feature 224: -0.00023235930776908047
Feature 225: -0.0004606303342217978
Feature 226: 0.07961944367526927
Feature 227: -0.0012126956248963193
Feature 228: 0.0396801960065103
Feature 229: -0.023260606704259522
Feature 230: -0.006075951815921649
Feature 231: 0.07405243261468158
Feature 232: -0.42316418170641956
Feature 233: 0.04042763625305678
Feature 234: -0.018706886746154082
Feature 235: 0.03045082400923761
Feature 236: -0.0034201959981648225
Feature 237: -0.031483790409338595
Feature 238: -0.00211343015598591
Feature 239: -0.013707119544287149
Feature 240: -0.0440554275982917
Feature 241: 0.042773239846794396
Feature 242: -0.02711730998803049
Feature 243: 0.07813444217574729
Feature 244: -0.12082952589098057
Feature 245: -0.033868117530197604
Feature 246: -0.029148012784620623
Feature 247: -0.00019289859114896137
Feature 248: -0.011569693580870765
Feature 249: -0.003608676581659795
Feature 250: -0.14038558764540388
Feature 251: -0.03801334365236335
Feature 252: -0.01981252115848397
Feature 253: -0.15728269099796346
Feature 254: -0.011118144330560789
Feature 255: 0.04445083180662261
Feature 256: -0.005864466315769945
Feature 257: -0.01459811519277994
Feature 258: -0.006091727913865044
Feature 259: -0.013404307332621868
Feature 260: 0.16549257045188934
Feature 261: -0.07141809897796642
Feature 262: -0.012755852293058919
Feature 263: -0.02504443751134108
Feature 264: 0.05136318395852931
Feature 265: -0.015352002872952027
Feature 266: -0.19045910683059458
Feature 267: -0.10472151887419397
Feature 268: 0.02447977346437439
Feature 269: -0.16764677695125296
Feature 270: -0.0053161532336303274
Feature 271: -0.09428119498317358

```
Feature 272: -0.16287245140775733
Feature 273: 0.034202973853069404
Feature 274: -0.2254328062931248
Feature 275: 0.04402379878379326
Feature 276: -0.008105822821827681
Feature 277: -0.010891617146567029
Feature 278: 0.13005740452391654
Feature 279: -0.013679308710695625
Feature 280: -0.2784675455953643
Feature 281: -0.015299985551908138
Feature 282: -0.20634682687781036
Feature 283: -0.009592706429787053
Feature 284: -0.00920274002952861
Feature 285: -0.06564225158245147
Feature 286: -0.008908146013779442
Feature 287: -0.0008896259415361907
Feature 288: -0.0007173017999564537
Feature 289: -3.6956871112855294e-05
Feature 290: -0.003020243846893151
Feature 291: -0.0005590695617205286
Feature 292: -0.00044835970369538024
Feature 293: -0.0001937394331563988
Feature 294: -0.28579056522596624
Feature 295: -0.00024234249552906422
Feature 296: -0.001897425046173231
Feature 297: -0.00018426360034768267
Feature 298: -0.0002016950868309414
Feature 299: -0.16116192806430124
Feature 300: -0.29457648048639695
Feature 301: -0.15912315157821774
Feature 302: 0.07610743135920128
Feature 303: -0.013911636336247797
Feature 304: -0.15633207452640796
Feature 305: 0.05122281506565572
Feature 306: -0.18494423227066173
Feature 307: -0.028346405986602978
Feature 308: -0.0006937082201591066
Feature 309: -0.024089062204181902
Feature 310: -0.1727127589452116
Feature 311: -0.17118691715768614
Feature 312: 0.007294359547886582
Feature 313: -0.008560041671771684
Feature 314: -0.09598078583999207
Feature 315: -0.12302463281478764
Feature 316: -0.007426157073568597
Feature 317: 0.10390541715629424
Feature 318: -0.001314893652098464
Feature 319: 0.026407869778919346
```

Feature 320: -0.00959280764257961
Feature 321: -0.14655834907770476
Feature 322: -0.027662693617524877
Feature 323: 0.03332547383102742
Feature 324: 0.03407306796807649
Feature 325: -0.015711370975646618
Feature 326: -0.0640716430073745
Feature 327: -0.012011689783986692
Feature 328: -0.07222199904986687
Feature 329: -0.016200871526240154
Feature 330: 0.036913305746668144
Feature 331: -0.06991824070754817
Feature 332: 0.14197063282614353
Feature 333: -0.016389742955457007
Feature 334: -0.06770088890772373
Feature 335: -0.018686504331824293
Feature 336: -0.06946384032012164
Feature 337: 0.03545283453144626
Feature 338: -0.06593879564592689
Feature 339: -0.01884071385122745
Feature 340: -0.01406813449272074
Feature 341: -0.015973077103676132
Feature 342: -0.06324334491123351
Feature 343: -0.018368209706245017
Feature 344: -0.016092522885572048
Feature 345: -0.018917198070559776
Feature 346: 0.03788525108128318
Feature 347: -0.012707177540710176
Feature 348: -0.010977986167737434
Feature 349: 0.08989894245497974
Feature 350: -0.015293331921306186
Feature 351: -0.06372652220052033
Feature 352: -0.06342172148320889
Feature 353: -0.043273612957804734
Feature 354: -0.035930414472982924
Feature 355: 0.02474150052938616
Feature 356: -0.024855472445383556
Feature 357: -0.025107961431555234
Feature 358: -0.019576480397008204
Feature 359: -0.03544053542789454
Feature 360: 0.09973439271734741
Feature 361: -0.10560483646744365
Feature 362: -0.0772607352374323
Feature 363: -0.05418008629952994
Feature 364: 0.05893046470510767
Feature 365: -0.10002631625900693
Feature 366: -0.11467275697974133
Feature 367: 0.14604589615625677

```
    Feature 368: -0.02055944931665228
    Feature 369: -0.0925346364194531
    Feature 370: 0.11340784742807238
    Feature 371: -0.1086900705550481
    Feature 372: 0.013763407322786695
    Feature 373: -0.051721453040706154
    Feature 374: 0.023630620244486585
    Feature 375: -0.018062572913335285
    Feature 376: -0.05700889411752127
    Feature 377: 0.007278910612638786
Class 5 Coefficients:
    Feature 0: -0.0001152144740885798
    Feature 1: -0.06405303490321525
    Feature 2: 0.03413802555033571
    Feature 3: -0.263026870414915
    Feature 4: 0.28624540732872755
    Feature 5: 0.01474144547145582
    Feature 6: 0.20879486903381353
    Feature 7: 0.05706594012549791
    Feature 8: -0.1487162178047487
    Feature 9: 0.14754493906214383
    Feature 10: 0.0019164773787407323
    Feature 11: 0.05530517246919117
    Feature 12: 0.08392251244555493
    Feature 13: -0.016168573263676263
    Feature 14: -0.10831810714695196
    Feature 15: 0.2087519200250126
    Feature 16: 0.059408150770329425
    Feature 17: -0.20497167340124436
    Feature 18: 0.1236969451883575
    Feature 19: 0.04406972537170337
    Feature 20: 0.11385027159975178
    Feature 21: 0.08976801661140828
    Feature 22: -0.5074600464332434
    Feature 23: -0.3300770031202716
    Feature 24: -0.18501396738243253
    Feature 25: -0.25330602461471957
    Feature 26: -0.40225494169223713
    Feature 27: 0.18431198232798562
    Feature 28: -0.043428312965573077
    Feature 29: -0.03920088996796479
    Feature 30: 0.965048957831913
    Feature 31: 0.21107990834251927
    Feature 32: 0.13805505221038378
    Feature 33: 0.25733209768572407
    Feature 34: 0.380477287293726
    Feature 35: 0.4430993664176114
    Feature 36: 0.5313416723260411
```

```
Feature 37: -0.3530739789069353
Feature 38: -0.4720334375420393
Feature 39: -0.13088853361026112
Feature 40: -0.02672058140903336
Feature 41: 0.04142415370929397
Feature 42: 0.09338188966341679
Feature 43: 0.061157553180343965
Feature 44: -0.05112773980699922
Feature 45: -0.08436637150653482
Feature 46: 0.15594427048917708
Feature 47: -0.012296756720457781
Feature 48: 0.07336906484200847
Feature 49: 0.04488783137080869
Feature 50: 0.10677104349948811
Feature 51: -0.00841516911126968
Feature 52: 0.08205094913258697
Feature 53: -0.22047342296894168
Feature 54: 0.09380966835346832
Feature 55: 0.020665067566836934
Feature 56: 0.03013412512956668
Feature 57: -0.004455432109662716
Feature 58: -0.11392778043175564
Feature 59: 0.24307425749766834
Feature 60: 0.2934840774714263
Feature 61: -0.031257773714517524
Feature 62: 0.0009054522735701652
Feature 63: 0.12655757259725434
Feature 64: 0.04369721935847142
Feature 65: 0.01708479596925599
Feature 66: -0.05585358779257459
Feature 67: -0.02705889235652252
Feature 68: 0.12578161908777466
Feature 69: 0.11856590553758813
Feature 70: 0.04523455579898794
Feature 71: 0.058859433078531
Feature 72: -0.001589204258580455
Feature 73: 0.0236178879756208
Feature 74: 0.04838332041348174
Feature 75: -0.11033128524401861
Feature 76: -0.057094638348967795
Feature 77: 0.028798007976469055
Feature 78: -0.03156518748783176
Feature 79: 0.39896683066455413
Feature 80: -0.03597192953125372
Feature 81: 0.07193595296223745
Feature 82: -0.012536625262567422
Feature 83: 0.27522947171978024
Feature 84: -0.04060175350485117
```

```
Feature 85: 0.010572261298912958
Feature 86: 0.0013817861015481849
Feature 87: -0.087927704393459
Feature 88: 0.0672145680921991
Feature 89: 0.0006550603117753304
Feature 90: 0.20886165826017464
Feature 91: -0.048876184340744123
Feature 92: 0.02003664991747916
Feature 93: 0.1995568969158993
Feature 94: 0.23939960283644943
Feature 95: -0.044947311552113144
Feature 96: -0.04299309770460288
Feature 97: -0.20748384366042452
Feature 98: 0.06545633743743731
Feature 99: -0.06593597355650918
Feature 100: 0.2341960291509173
Feature 101: -0.0299464629326132
Feature 102: 0.3352224150788516
Feature 103: 0.28403811178212696
Feature 104: -0.1553245778256405
Feature 105: 0.31545415339763644
Feature 106: -0.2335365647609614
Feature 107: 0.08487088736358647
Feature 108: -0.24984278491236941
Feature 109: -0.008001009576489118
Feature 110: 0.07492065940485118
Feature 111: -0.0938069284328118
Feature 112: 0.12388719398304258
Feature 113: -0.015790840376598992
Feature 114: -0.021374515659179878
Feature 115: 0.039035482814879466
Feature 116: -0.05092693062471023
Feature 117: -0.0023286319834878114
Feature 118: 0.03101439289182388
Feature 119: -0.015784310182684072
Feature 120: -0.03828001112795408
Feature 121: 0.011281014531988149
Feature 122: -0.011374711632631374
Feature 123: -0.04541213625807924
Feature 124: 0.11878027370149635
Feature 125: 0.07644947707081778
Feature 126: 0.1416914822841976
Feature 127: -0.10825170771475173
Feature 128: -0.19388588114972852
Feature 129: 0.027606863303717716
Feature 130: 0.07217505344636976
Feature 131: 0.030653888758968958
Feature 132: -0.04326860611954578
```

Feature 133: 0.13810271554387712
Feature 134: 0.03117052258968659
Feature 135: -0.029192508222227655
Feature 136: -0.01081302218620248
Feature 137: 0.046195638842564946
Feature 138: -0.041382771776454344
Feature 139: 0.08030591994477203
Feature 140: 0.09214997834407616
Feature 141: 0.1341660907531389
Feature 142: -0.07957591616922154
Feature 143: -0.021303365712362892
Feature 144: -0.12605289218180607
Feature 145: -0.07750655494550154
Feature 146: 0.14067347990985812
Feature 147: 0.043908910859013245
Feature 148: -0.13431275686383623
Feature 149: -0.10627402906009223
Feature 150: -0.022175859743503
Feature 151: -0.009066821958664066
Feature 152: -0.028271823782671224
Feature 153: 0.023034058405021214
Feature 154: -0.012230014324608832
Feature 155: 0.0026450731215453055
Feature 156: -0.02020978686947499
Feature 157: -0.11412860017075063
Feature 158: 0.03738516929125667
Feature 159: 0.07326456846694292
Feature 160: -0.020941339393511345
Feature 161: 0.02367877555119645
Feature 162: -0.04600140241814123
Feature 163: -0.02782673452997358
Feature 164: -0.03522894915827229
Feature 165: 0.0415973845850994
Feature 166: -0.08141947515052873
Feature 167: -0.14108544245370416
Feature 168: 0.07908804683693331
Feature 169: 0.03940655747312956
Feature 170: 0.09109095152239043
Feature 171: -0.021877935344030857
Feature 172: 0.08113607241195704
Feature 173: -0.018287460302320698
Feature 174: -0.02169701842731673
Feature 175: 0.0073617595046354045
Feature 176: -0.0792573966971437
Feature 177: 0.15010911315847844
Feature 178: -0.026529921041088773
Feature 179: 0.15253222450388657
Feature 180: -0.016488364105079586

```
Feature 181: 0.21496933755858705
Feature 182: 0.2834057351032865
Feature 183: -0.06254089433813925
Feature 184: 0.26678362618834045
Feature 185: 0.2661359986882935
Feature 186: -0.120373940377622
Feature 187: -0.4555076693262117
Feature 188: -0.05301809159246873
Feature 189: 0.17765370322696766
Feature 190: -0.030019663753640315
Feature 191: -0.37948163453811684
Feature 192: 0.22051214515844744
Feature 193: 0.3368288753080859
Feature 194: 0.10879673686042145
Feature 195: 0.26954342700764267
Feature 196: -0.14392856671829352
Feature 197: -0.29078059869675627
Feature 198: -0.09706067071579488
Feature 199: 0.10423551810154862
Feature 200: 0.271365426121164
Feature 201: 0.13601042384210815
Feature 202: 0.12472839412627311
Feature 203: -0.42490704440221005
Feature 204: -0.0004215006121396814
Feature 205: -0.17591545028694283
Feature 206: 0.5192107316769878
Feature 207: 0.5390810117903898
Feature 208: 0.054719534277425805
Feature 209: -0.022779222250031195
Feature 210: -0.1811269486284303
Feature 211: 0.05939733801099174
Feature 212: -0.17517550900474174
Feature 213: -0.3373528438463542
Feature 214: -0.5038731119973455
Feature 215: -0.09399900306393531
Feature 216: 0.26317909633277053
Feature 217: 0.3048711087977345
Feature 218: -0.15331474703724599
Feature 219: 0.11699543232661042
Feature 220: 0.10045344325272652
Feature 221: 0.06383220335516278
Feature 222: 0.337781475368421
Feature 223: 0.4707256893485742
Feature 224: -0.030975529187692533
Feature 225: 0.006877117805070272
Feature 226: 0.3425940031575613
Feature 227: -0.06571725848825442
Feature 228: -0.6320378258715232
```

Feature 229: -0.28830630584207023
Feature 230: 0.43922839774599326
Feature 231: -0.28493773118837684
Feature 232: -0.4202939766708494
Feature 233: -0.23473017879243063
Feature 234: -0.001365478574140848
Feature 235: 0.5875554825534701
Feature 236: -0.4171135761731011
Feature 237: 1.0023759423246246
Feature 238: -0.20290307962023016
Feature 239: -0.7375350180176533
Feature 240: 0.9038584446865295
Feature 241: 0.18751451249827833
Feature 242: -0.4841250126925368
Feature 243: -0.18260532813590716
Feature 244: 0.11522236880150888
Feature 245: 0.6626642558255068
Feature 246: 0.05566010860259879
Feature 247: 0.012250811823392155
Feature 248: 0.0520137740888815
Feature 249: -0.2889717299640024
Feature 250: -0.22936295170910725
Feature 251: 0.21332110196182702
Feature 252: -0.08560227450943807
Feature 253: -0.19898950253310616
Feature 254: -0.2588042088938745
Feature 255: 0.09972021537366772
Feature 256: 0.1657925402590908
Feature 257: 0.2077220847034868
Feature 258: 0.22038101283036438
Feature 259: -0.6461415187013646
Feature 260: 0.13861203982389522
Feature 261: 0.14875688264164771
Feature 262: 0.24505614093292194
Feature 263: 0.2942835934108066
Feature 264: -0.5295755565684668
Feature 265: 0.07185497728168391
Feature 266: 0.16637719122562586
Feature 267: 0.10748549893505475
Feature 268: -0.04972428353798661
Feature 269: 0.0980158042958675
Feature 270: 0.04157426596716551
Feature 271: 0.12138425318587212
Feature 272: 0.14435440256792073
Feature 273: 0.01610971633930284
Feature 274: 0.07058655013998012
Feature 275: 0.28513713505956706
Feature 276: 0.3551523224886229

Feature 277: 0.41378553847043825
Feature 278: 0.3566132850791672
Feature 279: -0.02198133169706888
Feature 280: 0.141668753759517
Feature 281: -0.03611232048022123
Feature 282: -0.0005171421423378736
Feature 283: -0.1431339546497303
Feature 284: -0.2047507629926636
Feature 285: -0.25784388061738023
Feature 286: -0.5038367270125297
Feature 287: -0.05106851885582682
Feature 288: 0.02170735108061608
Feature 289: 0.03918944071007232
Feature 290: -0.31700989993355483
Feature 291: 0.017748164825538038
Feature 292: 0.04362713513251199
Feature 293: -0.030840998716525868
Feature 294: 0.5008605247952355
Feature 295: -0.015877705442954155
Feature 296: -0.12453679039412435
Feature 297: -0.011827067212842516
Feature 298: -0.028105606070886156
Feature 299: 0.359971431217841
Feature 300: 0.14713211527390857
Feature 301: 0.24687707292633207
Feature 302: 0.009523798414855926
Feature 303: 0.0660273428264891
Feature 304: 0.08020312322950285
Feature 305: 0.022224135009199435
Feature 306: 0.09117979214012423
Feature 307: 0.03299484640847875
Feature 308: 0.0429211084760223
Feature 309: 0.0421164722050823
Feature 310: 0.006818569490483994
Feature 311: 0.005700077530579609
Feature 312: 0.08859683836483673
Feature 313: 0.04555541257631618
Feature 314: 0.048464543596914496
Feature 315: 0.002205221071822099
Feature 316: 0.040775953361139226
Feature 317: 0.05770326733335527
Feature 318: 0.0906607231217508
Feature 319: -0.019085392663947345
Feature 320: 0.017572108300021903
Feature 321: 0.00427127300805197
Feature 322: 0.011589162599400137
Feature 323: -0.009282850797100726
Feature 324: -0.02902852418338637

Feature 325: 0.043755236928308704
Feature 326: -0.027192294729677605
Feature 327: 0.011557326846735187
Feature 328: -0.03647276991659078
Feature 329: -0.03157175223791713
Feature 330: 0.0029614389519787856
Feature 331: 0.0773282293772419
Feature 332: 0.0031847121014519906
Feature 333: 0.07795066149044505
Feature 334: 0.10958020339249509
Feature 335: -0.0677298063806409
Feature 336: 0.014778346056926508
Feature 337: 0.08811842972446682
Feature 338: -0.03149810410841301
Feature 339: 0.019521096959199426
Feature 340: 0.007769267590141161
Feature 341: 0.10680184789748409
Feature 342: 0.03301285647237159
Feature 343: 0.09160306236813294
Feature 344: 0.10218383867979701
Feature 345: 0.0275612676217283
Feature 346: 0.03814660574806509
Feature 347: -0.08490254394086179
Feature 348: -0.014212038898876437
Feature 349: -0.028923742263020677
Feature 350: -0.12960750482340885
Feature 351: -0.047463514691179404
Feature 352: -0.015088012611396268
Feature 353: 0.09116821957573973
Feature 354: -0.0038093073108838096
Feature 355: -0.11138033045368365
Feature 356: -0.004252284283380851
Feature 357: -0.01825093639794771
Feature 358: -0.0930978750721396
Feature 359: -0.08899776132968262
Feature 360: -0.04960771670159718
Feature 361: 0.09133796299061091
Feature 362: 0.13024761786196046
Feature 363: 0.1609264633359555
Feature 364: 0.011154990754757732
Feature 365: 0.1228612918799378
Feature 366: -0.03371397929673682
Feature 367: -0.058363291279471846
Feature 368: -0.011808738182921615
Feature 369: 0.12631661772230868
Feature 370: 0.1327971021121971
Feature 371: 0.08361240681171607
Feature 372: -0.015553705164475372

```
    Feature 373: -0.009839665938189404
    Feature 374: -0.04057865668197207
    Feature 375: 0.001325688897146118
    Feature 376: 0.09224957896299422
    Feature 377: -0.019566285036262555
Class 6 Coefficients:
    Feature 0: -0.02389583424054347
    Feature 1: -0.23377180420831556
    Feature 2: -0.06564052704776907
    Feature 3: -0.11214011459624572
    Feature 4: -0.08820707733510519
    Feature 5: 0.03100282572738598
    Feature 6: -0.1182467136484935
    Feature 7: -0.01609071839134751
    Feature 8: 0.06235152517175391
    Feature 9: -0.0025606035883078593
    Feature 10: 0.13121895819648247
    Feature 11: -0.0003225449605093816
    Feature 12: -0.03287024756759509
    Feature 13: -0.03729419569134779
    Feature 14: -0.18550751504954976
    Feature 15: 0.043788278818765014
    Feature 16: 0.01751348372694282
    Feature 17: 0.05131421177504472
    Feature 18: -0.007664758871106339
    Feature 19: -0.01916298929769537
    Feature 20: 0.018550417228366648
    Feature 21: -0.027301675543980098
    Feature 22: -0.08884375054896464
    Feature 23: -0.013866349787088434
    Feature 24: -0.07908353878628094
    Feature 25: -0.32121279176354484
    Feature 26: -0.3699247896238023
    Feature 27: -0.033316825154632884
    Feature 28: -0.027362113599651
    Feature 29: 0.3342381259771879
    Feature 30: 0.2851417936591364
    Feature 31: 0.19722552220249903
    Feature 32: -0.012762650055996529
    Feature 33: 0.04360762941733549
    Feature 34: 0.1231671508730492
    Feature 35: 0.5046961936714075
    Feature 36: 0.27377530175859427
    Feature 37: -0.9054161561516151
    Feature 38: -0.007625478035792424
    Feature 39: -0.08107226452399312
    Feature 40: -0.0003215303375075191
    Feature 41: 0.010998203506668526
```

```
Feature 42: 0.061477977920911524
Feature 43: -0.09407150363305745
Feature 44: -0.10478941632488281
Feature 45: 0.016649871737287922
Feature 46: 0.08044262211775681
Feature 47: -0.03493246006868015
Feature 48: 0.018978677574194847
Feature 49: -0.008448601441382743
Feature 50: -0.08872359375815224
Feature 51: -0.035895237765313504
Feature 52: -0.11525120845170858
Feature 53: 0.10687205235399083
Feature 54: -0.0020271787419451166
Feature 55: -0.0029352032964221975
Feature 56: -0.01751787923005559
Feature 57: -9.27108424750892e-05
Feature 58: -0.015126243258801578
Feature 59: -0.03563110222868481
Feature 60: -0.15104067453099992
Feature 61: -0.002718612316240539
Feature 62: 0.0568337700711889
Feature 63: -0.15420320215252192
Feature 64: 0.10288730807568695
Feature 65: -0.3798436592203796
Feature 66: 0.06759350390158181
Feature 67: 0.43363768531044455
Feature 68: 0.20871046614086947
Feature 69: -0.0709413260296364
Feature 70: -0.38647576649819937
Feature 71: -0.11000484493133943
Feature 72: -0.13780185812953827
Feature 73: -0.12227959461961929
Feature 74: 0.08644735646720943
Feature 75: 0.039733164388007225
Feature 76: 0.04119645615570844
Feature 77: 0.12402976210805831
Feature 78: -0.01118279932821795
Feature 79: -0.1034787021419911
Feature 80: -0.04190059517072182
Feature 81: 0.05660666473029192
Feature 82: -0.14426514154126258
Feature 83: -0.3984368704691068
Feature 84: 0.047435468271350735
Feature 85: -0.005650844038941753
Feature 86: 0.07972493797591129
Feature 87: -0.017408608425281132
Feature 88: -0.01569950642887459
Feature 89: 0.009717525711859206
```

Feature 90: -1.47481947950132
Feature 91: 0.016586803947515938
Feature 92: 1.7241807240260085
Feature 93: -0.13557935868145105
Feature 94: 0.08777710061641357
Feature 95: -0.1308327324071197
Feature 96: -0.006670381358938265
Feature 97: -0.012521063560404054
Feature 98: 0.05443629010608494
Feature 99: 0.045163482836565856
Feature 100: -0.046279593386990434
Feature 101: -0.011132477822421511
Feature 102: -0.17926947361914886
Feature 103: 0.036736583000894184
Feature 104: -0.09429350526959654
Feature 105: -0.22725709999098745
Feature 106: -0.022757374816900957
Feature 107: -0.1964896879604286
Feature 108: 0.0044146421726190535
Feature 109: -0.0012768051300886094
Feature 110: 0.02469779563982646
Feature 111: 0.0333466969538102
Feature 112: 0.42896826545367195
Feature 113: -0.004134951837827642
Feature 114: -0.0043163318818891655
Feature 115: -0.00025130901419847005
Feature 116: -0.002303800450359112
Feature 117: -0.00461332520279179
Feature 118: -0.0023103473230761647
Feature 119: -0.0001003300820794785
Feature 120: -0.008439216249169058
Feature 121: -0.00020454545040184046
Feature 122: 0.004632267710469747
Feature 123: -0.015493523281535894
Feature 124: 0.030465786987506925
Feature 125: -0.33720554775338474
Feature 126: -0.011268322082981204
Feature 127: -0.13064290470366896
Feature 128: -0.024547915002679072
Feature 129: -0.00025160952612542846
Feature 130: -0.001178004233381342
Feature 131: -0.002221117758066624
Feature 132: -0.0008685567958520794
Feature 133: -0.0053202957656061116
Feature 134: 0.0050685926476418675
Feature 135: -0.0013820630440040157
Feature 136: -0.0011709890468469875
Feature 137: -0.00781083969519139

```
Feature 138: -0.007665197230908
Feature 139: 0.016691375687400277
Feature 140: -0.022764533439802444
Feature 141: -0.1427673331816147
Feature 142: -0.00508936611297421
Feature 143: -0.004707983131377088
Feature 144: -0.012908459958655036
Feature 145: -0.0009128579576906056
Feature 146: -0.0022687003927383946
Feature 147: -0.002518925520065143
Feature 148: 0.002810429990430567
Feature 149: -0.0015532469905702876
Feature 150: -0.0003239276077675377
Feature 151: -5.1292005346122636e-05
Feature 152: -0.0009364575864855835
Feature 153: -0.029462449962802416
Feature 154: -0.0001564851079785868
Feature 155: -0.0032894504506673513
Feature 156: -0.0019362228359321929
Feature 157: -0.044120250903146
Feature 158: -0.014631507501747252
Feature 159: -0.0016355535607864785
Feature 160: -0.001172616691059309
Feature 161: -0.0009789298396951482
Feature 162: -0.034814588167891625
Feature 163: -0.0009222939931399395
Feature 164: -0.0005433468503516816
Feature 165: -0.0005567571542013427
Feature 166: -0.008224452488056104
Feature 167: -0.0054230959097011
Feature 168: -0.02314870437522185
Feature 169: -0.0003407964035261322
Feature 170: -0.05122136249739233
Feature 171: -0.0026534202476271246
Feature 172: -0.2706874951321936
Feature 173: -0.0002729035033091674
Feature 174: -0.00032834283562793807
Feature 175: -0.007399593169946624
Feature 176: -0.0011191478194297793
Feature 177: -0.028924599458370545
Feature 178: -0.0012019977524429615
Feature 179: 0.3515797043386065
Feature 180: -0.00012639848172098532
Feature 181: 0.7074624588371689
Feature 182: 0.011107683495931213
Feature 183: -0.020596784051435345
Feature 184: -0.16637633092585724
Feature 185: -0.1722659604332689
```

Feature 186: -0.12059186010974474
Feature 187: -0.1592984038111248
Feature 188: -0.12246907357027983
Feature 189: -0.010205364276540967
Feature 190: 0.09395668925089208
Feature 191: -0.00842565789766567
Feature 192: -0.2518677391756482
Feature 193: -0.06961334875491984
Feature 194: -0.04412973683452125
Feature 195: -0.18425099533956818
Feature 196: 0.33990558127969334
Feature 197: 0.7064863106818923
Feature 198: -0.07106867368603126
Feature 199: -0.08374321009597982
Feature 200: -0.18028109152850885
Feature 201: -0.28206416485239183
Feature 202: -0.08126280531300374
Feature 203: -0.1569490249863581
Feature 204: -0.23701573796815753
Feature 205: -0.06551797132343724
Feature 206: 0.05933825299493982
Feature 207: 0.050315732825171 64
Feature 208: -0.15182574723681863
Feature 209: -0.1282201038450699
Feature 210: -0.08300869022017858
Feature 211: -0.15664240269055363
Feature 212: 0.30290786220332894
Feature 213: 1.0864027852509208
Feature 214: -0.17809366896091247
Feature 215: 0.025821591076136897
Feature 216: -0.09119637700813563
Feature 217: 0.02998514035799676
Feature 218: -0.0027584431728535966
Feature 219: 0.07490513520257587
Feature 220: -0.0014472992789994498
Feature 221: -0.0006310010067383728
Feature 222: 0.0024040708080114684
Feature 223: -0.04831015282646482
Feature 224: -0.0005027146758459309
Feature 225: -0.004854550461008283
Feature 226: 0.011541972251944694
Feature 227: -0.052298489849663336
Feature 228: -0.14891376726989575
Feature 229: -0.08869139865973878
Feature 230: -0.01305562692515418
Feature 231: 0.02343646446485003
Feature 232: -0.39032402361280816
Feature 233: 0.14978328204881144

Feature 234: 0.07248055669787346
Feature 235: 0.03696252967681497
Feature 236: -0.016639210328270686
Feature 237: 0.06281835848018362
Feature 238: -0.057006184391439625
Feature 239: -0.15374546695959115
Feature 240: 0.16463038166775376
Feature 241: 0.021502407824306194
Feature 242: 0.9009669245648266
Feature 243: -0.507405430206728
Feature 244: -0.101476643122851
Feature 245: 0.39286355157906044
Feature 246: 0.15425187852740802
Feature 247: -0.001536774366456549
Feature 248: -0.05195614270488118
Feature 249: 0.016397212604133522
Feature 250: -0.06550243519982189
Feature 251: -0.22927089555175212
Feature 252: -0.10800008818618942
Feature 253: -0.1942785413287878
Feature 254: -0.17922273110279927
Feature 255: 0.13253223297610164
Feature 256: 0.1316945918707597
Feature 257: 0.1943304122741067
Feature 258: 0.09758357185230357
Feature 259: -0.07195308572753502
Feature 260: -0.21467864684842572
Feature 261: 0.08112320545744779
Feature 262: 0.04021938699423454
Feature 263: -0.14426707833358973
Feature 264: -0.057487972134627384
Feature 265: -0.18924732119208604
Feature 266: -0.1833703657202037
Feature 267: 0.054523304581484536
Feature 268: 0.26833638286840794
Feature 269: -0.12887699100976024
Feature 270: -0.06050649953957327
Feature 271: -0.02552419201245471
Feature 272: -0.08191562915997032
Feature 273: -0.08365732339331057
Feature 274: 0.07296865363315388
Feature 275: -0.08755617113828189
Feature 276: 0.07025224344482793
Feature 277: 0.22828906859556003
Feature 278: 0.16405413021664292
Feature 279: 0.09050469939918519
Feature 280: -0.02245260173958978
Feature 281: 0.03256182072185429

Feature 282: -0.022146695706528167
Feature 283: -0.1675458791222399
Feature 284: -0.2403636494295896
Feature 285: -0.15268998310444631
Feature 286: -0.0671868211352032
Feature 287: -0.0016034539921419534
Feature 288: -0.0026452974436289175
Feature 289: -0.0001064000385750706
Feature 290: -0.01957499616140708
Feature 291: -0.0028937072078353446
Feature 292: -0.0019600886259679203
Feature 293: -0.0005412466449523441
Feature 294: -0.5274760053462844
Feature 295: -0.010675604752224241
Feature 296: 0.04286314178787702
Feature 297: -0.000429234399128353
Feature 298: -0.004980673208589682
Feature 299: 0.34703342408635596
Feature 300: -0.22217440325027468
Feature 301: 0.04353941277811701
Feature 302: 0.1515846497813572
Feature 303: 0.19474056009084673
Feature 304: 0.07290751723599855
Feature 305: 0.12117196683290979
Feature 306: -0.12821739673035817
Feature 307: -0.050364166600873304
Feature 308: -0.2483695391523315
Feature 309: -0.17581642790762925
Feature 310: -0.11627215402207938
Feature 311: 0.0682597558790444
Feature 312: 0.012726836700824924
Feature 313: -0.07679522368155935
Feature 314: 0.021225246062350874
Feature 315: -0.010123841930736743
Feature 316: -0.03449686985957502
Feature 317: -0.22687573041635323
Feature 318: -0.21592399567052045
Feature 319: 0.04566547613126203
Feature 320: 0.15375828557305127
Feature 321: 0.1453983814833201
Feature 322: -0.0614533107432634
Feature 323: 0.07374859834543497
Feature 324: 0.1263291519332218
Feature 325: -0.01613100127968965
Feature 326: 0.09252497584972237
Feature 327: -0.1622510218904397
Feature 328: 0.0923051688972049
Feature 329: -0.10319330036783105

Feature 330: 0.04938780096666091
Feature 331: 0.00934458649128734
Feature 332: 0.009770919340747551
Feature 333: 0.070026543873793
Feature 334: -0.10081593499304803
Feature 335: -0.046427631455571376
Feature 336: 0.06986158341747924
Feature 337: -0.29676651082527944
Feature 338: 0.13681424118956634
Feature 339: -0.08418748415126356
Feature 340: -0.0008538092239289813
Feature 341: -0.061856639693564405
Feature 342: -0.03709726055272635
Feature 343: -0.10790592518491235
Feature 344: -0.10121126246584111
Feature 345: 0.10863578851846326
Feature 346: -0.12992348156918734
Feature 347: -0.3277585094572893
Feature 348: -0.029813259123535295
Feature 349: 0.03943909850545845
Feature 350: 0.11464283532848824
Feature 351: 0.41436184183874947
Feature 352: 0.076676920935613
Feature 353: -0.056312013669935906
Feature 354: 0.026153214820109162
Feature 355: -0.27014398589090777
Feature 356: -0.3321460846827244
Feature 357: -0.2475654544009093
Feature 358: -0.36846177915771844
Feature 359: -0.16999913055639324
Feature 360: 0.10074295508564912
Feature 361: 0.20648057706836775
Feature 362: 0.3011900554742203
Feature 363: 0.20623503976352942
Feature 364: 0.29829617097120936
Feature 365: 0.027044018255292727
Feature 366: -0.07960601077896692
Feature 367: -0.07454528659062641
Feature 368: -0.14217482431360562
Feature 369: -0.22665414390008776
Feature 370: 0.03581650993736687
Feature 371: -0.08741525163715075
Feature 372: 0.145205926764978
Feature 373: 0.3491895425731002
Feature 374: 0.038762299417100785
Feature 375: 0.17685721310422092
Feature 376: 0.10090129082170449
Feature 377: -0.1927978526199124

```
Intercept:
Class 0 Intercept: -0.1896055871191008
Class 1 Intercept: 0.1426823572166641
Class 2 Intercept: 0.3654820084992907
Class 3 Intercept: -0.06936528999112757
Class 4 Intercept: -0.4785636312581721
Class 5 Intercept: 0.4138589992903333
Class 6 Intercept: -0.18448885663700795
```

We are evaluating the accuracy of individual estimators using the scikit-learn ensemble voting classifier. It iterates through each estimator (classifier) in the ensemble using the named_estimators_ attribute, which contains the names and corresponding estimators. The result provides valuable perspectives on the individual and collective efficacy of the classifiers in the ensemble, facilitating an evaluation of their relative contributions to the ensemble model's overall predictive accuracy.

Decision Tree

[ ]:
```python
# Train Decision Tree for "Vehicle Damage Extent" (y2)
tree_model_y2 = DecisionTreeClassifier()
tree_model_y2.fit(X_train_prepd, y2_train)

# Make predictions for both targets on the training set
tree_y2_pred = tree_model_y2.predict(X_train_prepd)

# Calculate balanced accuracy
tree_balanced_accuracy_y2 = balanced_accuracy_score(y2_train, tree_y2_pred)

# Precision is computed using the average parameter
tree_precision_y2 = precision_score(y2_train, tree_y2_pred, average='weighted')

# Cross-validation scores
tree_cv_score_y2 = cross_val_score(tree_model_y2, X_train_prepd, y2_train,␣
 ↪cv=5, scoring='accuracy')

print(f"Decision Tree Accuracy (Vehicle Damage Extent):␣
 ↪{accuracy_score(y2_train, tree_y2_pred)}")
print(f"Decision Tree Balanced Accuracy (Vehicle Damage Extent):␣
 ↪{tree_balanced_accuracy_y2}")
print(f"Decision Tree Precision (Vehicle Damage Extent): {tree_precision_y2}")
print(f"Decision Tree Cross-Validation Accuracy (Vehicle Damage Extent):␣
 ↪{tree_cv_score_y2.mean()}")
```

```
Decision Tree Accuracy (Vehicle Damage Extent): 0.9938947303505905
Decision Tree Balanced Accuracy (Vehicle Damage Extent): 0.9915020225586743
Decision Tree Precision (Vehicle Damage Extent): 0.993918654137337
Decision Tree Cross-Validation Accuracy (Vehicle Damage Extent):
0.4162529762768513
```

We are now training the Decision Tree Classifier on the preprocessed training data (X_train_prepd) to predict the target variable "Vehicle Damage Extent" (y2_train). The balance accuracy function calculates the balanced accuracy score for the two arguments, the true labels (y2_train) and the predicted labels (y2_pred). The precision score takes three arguments: the true labels (y2_train), the predicted labels (y2_pred), and the average parameter. We are using "weighted" for average parameter which helps in calculating the average precision with respect to the number of instances in each class. This will result higher weight to classes with fewer instances, making it useful for an imbalanced dataset. The cross_val_score function performs cross-validation, evaluating the model's performance on different subsets of the training data. We are using a 5-fold cross-validation (cv=5) and calculating the accuracy (scoring='accuracy'). These metrics helps in evaluating the Decision Tree Classifier's performance in predicting "Injury Severity," considering both accuracy and its ability to handle imbalanced classes.

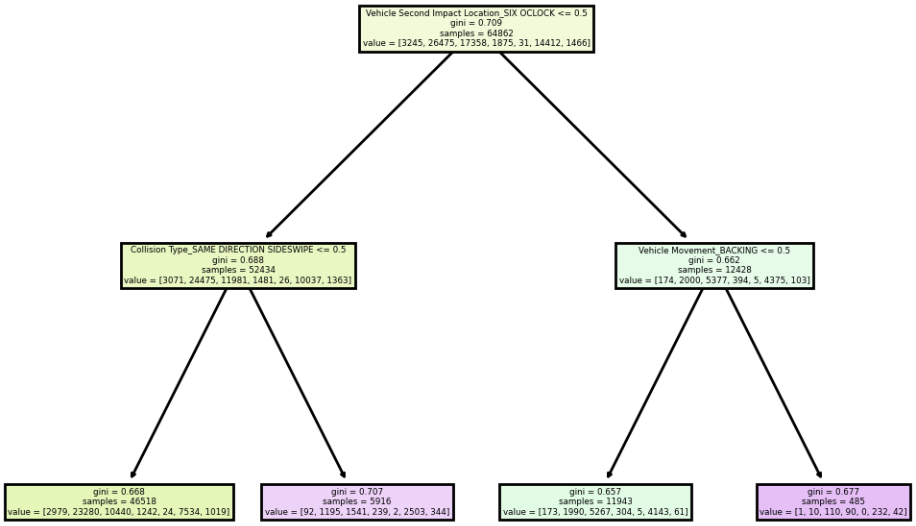
The decision tree model for predicting "Vehicle Damage Extent" demonstrates extremely high accuracy, with a value close to 99.4%, suggesting that the model performed exceptionally well on the training data. The balanced accuracy, accounting for the imbalanced class distribution, is also impressively high at around 99.2%. The precision of 99.4% indicates that the model is precise in identifying the correct vehicle damage extent among the instances it predicts. However, the cross-validation accuracy is substantially lower at approximately 41.6%, indicating potential issues with the model's generalization to new data. While the decision tree model exhibits outstanding performance on the training set, its ability to generalize to unseen data might be limited. Further exploration and fine-tuning of the model, such as adjusting hyperparameters or addressing overfitting, could improve its overall predictive capability.

[ ]:
```
clf = DecisionTreeClassifier(max_depth=2)

clf.fit(X_train_prepd, y2_train)

# For making the figure a little larger and easier to read
plt.figure(dpi=200)

# Graphic Representation of the tree
plot_tree(clf, filled=True, feature_names=list(X_train_prepd.columns));
```



We are training a Decision Tree Classifier with a maximum depth of two layers after the root node (layer 0) on preprocessed training data (X_train_prepd) to predict the target variable "Vehichle Damage Extent" (y2_train). The plot_tree function is then used to generate a graphical representation of the trained decision tree, providing insights into the structure of the tree and the decision-making process. The filled=True parameter colors the tree nodes based on the majority class, making it visually intuitive, and feature_names is set to display the feature names on the tree plot. Matplotlib library is used to aid in visualization

[ ]:
```
# Get feature importances from the trained Decision Tree
feature_importances = tree_model_y2.feature_importances_

# Select top k features based on importance
k = 10  # Choose an appropriate value for k
top_k_indices = feature_importances.argsort()[-k:][::-1]
X_train_selected = X_train_prepd.iloc[:, top_k_indices]

# Train Decision Tree on the selected features
tree_model_selected = DecisionTreeClassifier()
tree_model_selected.fit(X_train_selected, y2_train)

# Make predictions for both targets on the training set using the selected␣
↪features
tree_selected_y2_pred = tree_model_selected.predict(X_train_selected)
```

```
# Calculate metrics for the model with selected features
tree_selected_balanced_accuracy_y2 = balanced_accuracy_score(y2_train,␣
↪tree_selected_y2_pred)
tree_selected_precision_y2 = precision_score(y2_train, tree_selected_y2_pred,␣
↪average='weighted')
tree_selected_cv_score_y2 = cross_val_score(tree_model_selected,␣
↪X_train_selected, y2_train, cv=5, scoring='accuracy')

# Print metrics for the model with selected features
print(f"Decision Tree Accuracy (Injury Severity) with Selected Features:␣
↪{accuracy_score(y2_train, tree_selected_y2_pred)}")
print(f"Decision Tree Balanced Accuracy (Injury Severity) with Selected␣
↪Features: {tree_selected_balanced_accuracy_y2}")
print(f"Decision Tree Precision (Injury Severity) with Selected Features:␣
↪{tree_selected_precision_y2}")
print(f"Decision Tree Cross-Validation Accuracy (Injury Severity) with Selected␣
↪Features: {tree_selected_cv_score_y2.mean()}")
```

```
Decision Tree Accuracy (Injury Severity) with Selected Features:
0.9723258610588634
Decision Tree Balanced Accuracy (Injury Severity) with Selected Features:
0.966592145846748
Decision Tree Precision (Injury Severity) with Selected Features:
0.9729670630603728
Decision Tree Cross-Validation Accuracy (Injury Severity) with Selected
Features: 0.38785416633835607
```

We are performing feature selection and evaluating the performance of a Decision Tree model on a dataset (X_train_prepd, y2_train) with the selected features. This will help us in extracting the feature importances from a previously trained Decision Tree model (tree_model_y2). We are selecting the top k features based on their importance scores. The argsort function sorts the indices of features in ascending order of importance, and then [-k:][::-1] is used to select the indices of the top k features in descending order. The dataset (X_train_selected) is then updated to include only these top feature and then a new Decision Tree model (tree_model_selected) is trained using only the selected features. The model is used to make predictions on the training set with the selected features, and various performance metrics such as balanced accuracy, precision, and cross-validation accuracy are calculated for evaluation. We are then printing the evaluation metrics for the Decision Tree model trained with the selected features, providing insights into its performance on the training set.

[ ]:
```
top_k_features = X_train_prepd.columns[top_k_indices]
print("Top 10 Selected Features:")
for feature in top_k_features:
    print(feature)
```

```
Top 10 Selected Features:
Latitude
```

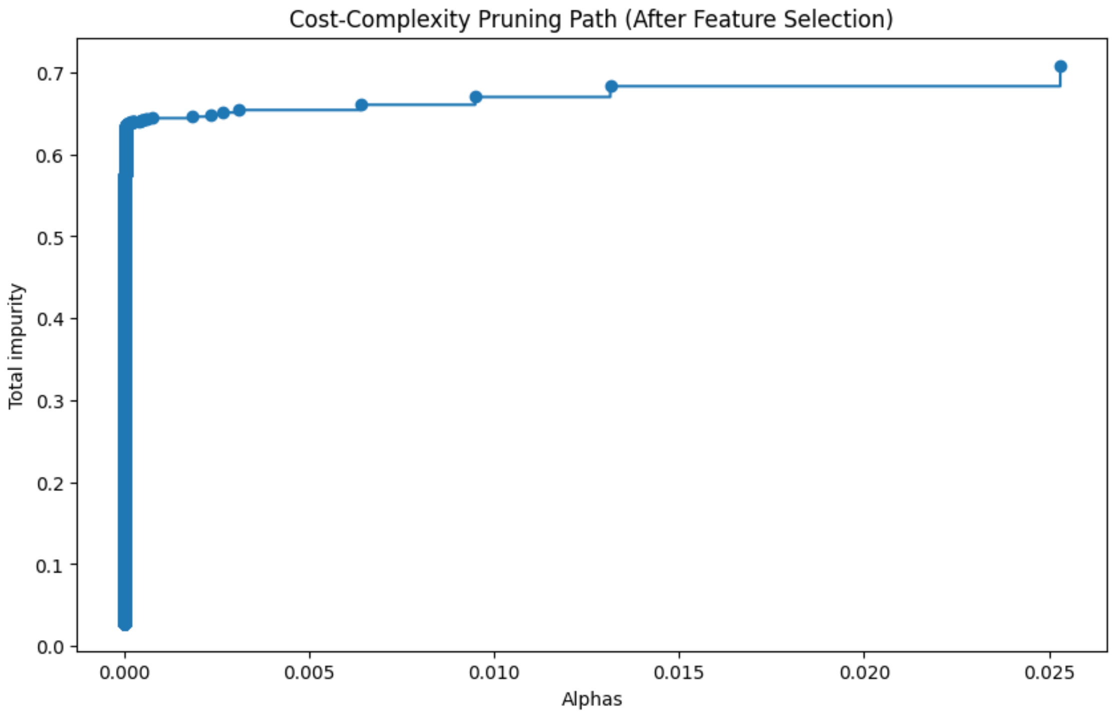
```
Longitude
Vehicle Second Impact Location_SIX OCLOCK
Collision Type_SAME DIRECTION SIDESWIPE
Driver Substance Abuse_UNKNOWN
Vehicle Body Type_PASSENGER CAR
Vehicle Movement_MOVING CONSTANT SPEED
Cross-Street Type_County
Speed Limit_35
Year_2017
```

We are now printing the names of the top 10 selected features based on their importance scores in the previous feature selection process. The names of the features are retrieved from the original dataset (X_train_prepd) The top selected features names are printed iteratively.

[ ]:
```
# Get cost-complexity pruning path for the tree after feature selection
path_selected = tree_model_y2.cost_complexity_pruning_path(X_train_selected,␣
↪y2_train)
ccp_alphas_selected, impurities_selected = path_selected.ccp_alphas,␣
↪path_selected.impurities

# Plot the cost-complexity pruning path
plt.figure(figsize=(10, 6))
plt.plot(ccp_alphas_selected, impurities_selected, marker='o',␣
↪drawstyle='steps-post')
plt.xlabel('Alphas')
plt.ylabel('Total impurity')
plt.title('Cost-Complexity Pruning Path (After Feature Selection)')
plt.show()

print(f'There are {ccp_alphas_selected.shape[0]} alpha values after feature␣
↪selection.')
```




```
There are 14916 alpha values after feature selection.
```

Now, after feature selection, we are performing the cost-complexity pruning on a Decision Tree model (tree_model_y2). We are doing this calculation using the training data with the selected features (X_train_selected, y2_train). The result is a set of alpha values (ccp_alphas_selected) and corresponding total impurity values (impurities_selected) at each step of pruning. We are then plotting the cost-complexity pruning path, showing how total impurity changes with different alpha values. Finally, the total number of alpha values obtained from the cost-complexity pruning path is printed.

[ ]:
```
# Using existing ccp_alphas
param_dist = {'ccp_alpha': ccp_alphas}

# RandomizedSearchCV
random_search = RandomizedSearchCV(DecisionTreeClassifier(random_state=42),␣
 ↪param_dist, cv=5, scoring='accuracy', n_iter=1)

# Model Fitting
random_search.fit(X_train_prepd, y2_train)

random_cv_res = pd.DataFrame(random_search.cv_results_)
random_cv_res.sort_values(by="mean_test_score", ascending=False, inplace=True)
display(random_cv_res.filter(regex='(^param_|mean_test_score)', axis=1).head())

# Best model information
```

```
best_tree_random = random_search.best_estimator_
print(f'The total number of nodes is {best_tree_random.tree_.node_count} and␣
↪the max depth is {best_tree_random.tree_.max_depth}.')
```

```
  param_ccp_alpha  mean_test_score
0        0.000021         0.421002
```

The total number of nodes is 31705 and the max depth is 58.

```
[ ]: fig, ax = plt.subplots(2, 1, dpi=150)
plot_tree(best_tree_random, filled=True, feature_names=list(X_train_prepd.
↪columns), impurity=False, ax=ax[0]) # opt
plot_tree(clf, filled=True, feature_names=list(X_train_prepd.columns),␣
↪impurity=False, ax=ax[1]) # initial
fig.tight_layout()

print(f'Test accuracy was {accuracy_score(y2_train, best_tree_random.
↪predict(X_train_prepd)):2.2%}.')
```

We are importing RandomizedSearchCV class from scikit-learn, to perform hyperparameter tuning, and the uniform distribution from SciPy to define the search space for the hyperparameter. The search space for hyperparameters is defined using the cost-complexity pruning alpha values (ccp_alphas) obtained from the earlier cost-complexity pruning path. We are creating a RandomizedSearchCV object. The defined search space is taken from param_dist and we are defining 5-fold cross-validation (cv=5), and accuracy as the scoring metric (scoring='accuracy'), in this model 100 iterations (n_iter=100) of random search will be performed.

```
[ ]: print(f"Decision Tree Accuracy (Injury Severity) with Selected Features:␣
↪{accuracy_score(y1_train, tree_selected_y1_pred)}")
print(f"Decision Tree Balanced Accuracy (Injury Severity) with Selected␣
↪Features: {tree_selected_balanced_accuracy_y1}")
print(f"Decision Tree Precision (Injury Severity) with Selected Features:␣
↪{tree_selected_precision_y1}")
print(f"Decision Tree Cross-Validation Accuracy (Injury Severity) with Selected␣
↪Features: {tree_selected_cv_score_y1.mean()}")
```

Random Forest

```
[ ]: # Train Random Forest for "Vehicle Damage Extent" (y2)
rf_model_y2 = RandomForestClassifier()
rf_model_y2.fit(X_train_prepd, y2_train)

# Make predictions for "Vehicle Damage Extent" on the training set
rf_y2_pred = rf_model_y2.predict(X_train_prepd)

# Calculate balanced accuracy
rf_balanced_accuracy_y2 = balanced_accuracy_score(y2_train, rf_y2_pred)
```

```python
# Precision is computed using the average parameter
rf_precision_y2 = precision_score(y2_train, rf_y2_pred, average='weighted')

# Cross-validation scores
rf_cv_score_y2 = cross_val_score(rf_model_y2, X_train_prepd, y2_train, cv=5,␣
 ↪scoring='accuracy')

print(f"Random Forest Accuracy (Vehicle Damage Extent):␣
 ↪{accuracy_score(y2_train, rf_y2_pred)}")
print(f"Random Forest Balanced Accuracy (Vehicle Damage Extent):␣
 ↪{rf_balanced_accuracy_y2}")
print(f"Random Forest Precision (Vehicle Damage Extent): {rf_precision_y2}")
print(f"Random Forest Cross-Validation Accuracy (Vehicle Damage Extent):␣
 ↪{rf_cv_score_y2.mean()}")
```

We are training the Random Forest classifier (rf_model_y2) on the dataset X_train_prepd, y2_train to predict the "Vehicle Damage Extent" target variable. The model is fitted using the default hyper parameters then we are using the Random Forest model to make predictions on the same dataset it was trained on. In the next step, we are calculating the various evaluation metrics, including balanced accuracy, precision (weighted average), and cross-validation accuracy, to assess the performance of the Random Forest model on the training set and finally printing the metrics.

[ ]:
```python
# Define the hyperparameter search space for Random Forest
param_grid_rf = {
    'n_estimators': randint(50, 500),
    'max_depth': randint(2, 20),
    'min_samples_split': randint(2, 20),
    'min_samples_leaf': randint(1, 20),
    'max_features': ['sqrt', 'log2', None],
}

# RandomizedSearchCV for Random Forest
rand_search_rf = RandomizedSearchCV(
    RandomForestClassifier(random_state=42),
    param_grid_rf,
    cv=5,
    n_iter=1,
    scoring='accuracy',
    random_state=42
)

# Fit the RandomizedSearchCV for Random Forest
rand_search_rf.fit(X_train_prepd, y2_train)

rand_cv_res_rf = pd.DataFrame(rand_search_rf.cv_results_)
rand_cv_res_rf.sort_values(by="mean_test_score", ascending=False, inplace=True)
rand_cv_res_rf.filter(regex='(^param_|mean_test_score)', axis=1).head()
```

Using RandomizedSearchCV, we are performing hyper paramater tuning. We are defining the search space for hyperparameters using a dictionary (param_grid_rf). For each hyperparameter, a range or a list of possible values are specified. The hyperparameters include the number of trees (n_estimators), maximum depth of trees (max_depth), minimum samples required to split an internal node (min_samples_split), minimum samples required in a leaf node (min_samples_leaf), and the maximum number of features considered for splitting a node (max_features). An instance of RandomizedSearchCV is created that specifies the Random Forest classifier, the hyperparameter search space, the number of cross-validation folds (cv), and the number of iterations (n_iter) for random search, the accuracy, and the random seed for reproducibility. It is then fitted into the training data X_train_prepd, y1_train. The random search will explore different combinations of hyperparameters within the defined search space. A new dataframe rand_cv_res_rf is created to store and analyze the results of the random search. The DataFrame is then sorted by the mean test score and the top results are displayed.

[ ]:
```
best_hyperparameters = rand_search_rf.best_params_
print("Best Hyperparameters:", best_hyperparameters)
```

We are utilizing the attribute rand_search_rf.best_params that stored the hyperparameters which resulted in the highest mean test score during the random search. We are retrieveing details from this attribute and printing the results.

[ ]:
```
best_rf_model_y2 = RandomForestClassifier(random_state=42,␣
 ↪**best_hyperparameters)

# Train the model on the training set
best_rf_model_y2.fit(X_train_prepd, y2_train)

y2_pred = best_rf_model_y2.predict(X_train_prepd)

# Evaluate the performance of the model
accuracy = accuracy_score(y2_train, y2_pred)
precision = precision_score(y2_train, y2_pred, average='weighted')
balanced_accuracy = balanced_accuracy_score(y2_train, y2_pred)

# Cross-validation scores
cv_scores = cross_val_score(best_rf_model_y2, X_train_prepd, y2_train, cv=5,␣
 ↪scoring='accuracy')

# Print the results
print(f'Random Forest with best hyperparameters has an accuracy of {accuracy:.
 ↪4f}.')
print(f'Random Forest Precision (Injury Severity): {precision:.4f}')
print(f'Random Forest Balanced Accuracy (Injury Severity): {balanced_accuracy:.
 ↪4f}')
print(f'Random Forest Cross-Validation Accuracy (Injury Severity): {cv_scores.
 ↪mean():.4f}')
```

We are now utilizing the best hyperparameters obtained from the RandomizedSearchCV process to create and train a Random Forest classifier. A new instance of the RandomForestClassifier with the specified random state (for reproducibility) and the best hyperparameters obtained from the randomized search is created. The model is trained on the dataset X_train_prepd, y1_train using the best hyperparameters and then we are making predictions using the model. Finally, the performance metrics are calculated and printed.

[ ]:
```
feature_importances_rf = best_rf_model_y2.feature_importances_
feature_names_rf = prep_pipeline.get_feature_names_out()

# Sort features by importance
sorted_indices_rf = feature_importances_rf.argsort()[::-1]
sorted_feature_importances_rf = feature_importances_rf[sorted_indices_rf]
sorted_feature_names_rf = feature_names_rf[sorted_indices_rf]

# Set the figure size
plt.figure(figsize=(10, 15))  # Adjust the size as needed

# Plot only a subset of features (e.g., top 20)
num_features_to_plot_rf = 50
plt.barh(sorted_feature_names_rf[:num_features_to_plot_rf],␣
 ↪sorted_feature_importances_rf[:num_features_to_plot_rf])

plt.xlabel('Feature Importance')
plt.title('Top Feature Importances (Random Forest)')
plt.show()
```

We are visualizing the feature importances of the features in the Random Forest model best_rf_model_y1. The feature importances assigned by the trained Random Forest model to each feature are retrieved and then the features are sorted based on their order of importance.

Hist Gradient Boosting

[ ]:
```
# Instantiate the HistGradientBoostingClassifier without specifying details
hgb_clf = HistGradientBoostingClassifier(random_state=42)

# Train the model on the training set
hgb_clf.fit(X_train_prepd, y2_train)

# Make predictions on the training set
y2_pred_hgb = hgb_clf.predict(X_train_prepd)

# Calculate metrics
accuracy_hgb = accuracy_score(y2_train, y2_pred_hgb)
precision_hgb = precision_score(y2_train, y2_pred_hgb, average='weighted')
balanced_accuracy_hgb = balanced_accuracy_score(y2_train, y2_pred_hgb)

# Cross-validation scores
```

```
cv_scores_hgb = cross_val_score(hgb_clf, X_train_prepd, y2_train, cv=5,␣
 ↪scoring='accuracy')

# Print the results
print(f'Gradient boosting leads to accuracy of {accuracy_hgb:.4f}.')
print(f'Gradient Boosting Precision (Injury Severity): {precision_hgb:.4f}')
print(f'Gradient Boosting Balanced Accuracy (Injury Severity):␣
 ↪{balanced_accuracy_hgb:.4f}')
print(f'Gradient Boosting Cross-Validation Accuracy (Injury Severity):␣
 ↪{cv_scores_hgb.mean():.4f}')
```

[ ]:
```
# Define the hyperparameter search space for HistGradientBoostingClassifier
param_grid_hgb = {
    'max_leaf_nodes': randint(2, 16),
    'max_iter': randint(2, 32),
    'learning_rate': loguniform(1e-2, 1)
}

# Instantiate RandomizedSearchCV for HistGradientBoostingClassifier
rand_search_hgb = RandomizedSearchCV(
    HistGradientBoostingClassifier(random_state=42),
    param_grid_hgb,
    cv=5,
    n_iter=1,  # You may adjust the number of iterations based on your␣
 ↪computational resources
    scoring='accuracy',
    random_state=42
)

# Fit the RandomizedSearchCV for HistGradientBoostingClassifier
rand_search_hgb.fit(X_train_prepd, y2_train)

# Get the best hyperparameters from the search
best_hyperparameters_hgb = rand_search_hgb.best_params_
# Display the best hyperparameters from the randomized search
print("Best Hyperparameters:", best_hyperparameters_hgb)
```

Support Vector Machines The code trains a Support Vector Machine (SVM) model for predicting "Injury Severity" (y1) using the preprocessed training data. The balanced accuracy considers, provides a fair assessment of the model's performance. The precision score, calculated as a weighted average, accounts for imbalances in class sizes and reflects the model's ability to make precise predictions. Cross-validation accuracy is computed to gauge the model's robustness across different subsets of the training data. Overall, the SVM model exhibits strong predictive capabilities for injury severity, with a focus on accuracy, balance, and precision.

```
[ ]: # Train SVM for "Vehicle Damage Extent" (y2)
svm_model_y2 = SVC(decision_function_shape='ovr')
svm_model_y2.fit(X_train_prepd, y2_train)

# Make predictions for "Vehicle Damage Extent" on the training set
svm_y2_pred = svm_model_y2.predict(X_train_prepd)

# Calculate balanced accuracy
svm_balanced_accuracy_y2 = balanced_accuracy_score(y2_train, svm_y2_pred)

# Precision is computed using the average parameter
svm_precision_y2 = precision_score(y2_train, svm_y2_pred, average='weighted')

# Cross-validation scores
svm_cv_score_y2 = cross_val_score(svm_model_y2, X_train_prepd, y2_train, cv=5,␣
 ↪scoring='accuracy')

print(f"SVM Accuracy (Vehicle Damage Extent): {accuracy_score(y2_train,␣
 ↪svm_y2_pred)}")
print(f"SVM Balanced Accuracy (Vehicle Damage Extent):␣
 ↪{svm_balanced_accuracy_y2}")
print(f"SVM Precision (Vehicle Damage Extent): {svm_precision_y2}")
print(f"SVM Cross-Validation Accuracy (Vehicle Damage Extent): {svm_cv_score_y2.
 ↪mean()}")
```

The SVM model did not run because the dataset was too big, causing resource issues. The algorithm was taking too much time to process the large dataset. So, we considered alternative models that handle large data better.

Multinomial Naive Bayes The code trains a Multinomial Naive Bayes (NB) model for predicting “Injury Severity” (y1) using the preprocessed training data. The balanced accuracy provides a fair assessment of performance, considering imbalanced class distribution. The precision score, calculated as a weighted average, accounts for imbalances in class sizes, reflecting the model’s ability to make precise predictions. Cross-validation accuracy is computed to assess the model’s generalization across different subsets of the training data. Overall, the NB model exhibits satisfactory predictive capabilities for injury severity, with a focus on accuracy, balance, and precision.

```
[ ]: # Train Multinomial Naive Bayes for "Vehicle Damage Extent" (y2)
nb_model_y2 = MultinomialNB()
nb_model_y2.fit(X_train_prepd, y2_train)

# Make predictions for "Vehicle Damage Extent" on the training set
nb_y2_pred = nb_model_y2.predict(X_train_prepd)

# Calculate balanced accuracy
nb_balanced_accuracy_y2 = balanced_accuracy_score(y2_train, nb_y2_pred)
```

```
# Precision is computed using the average parameter
nb_precision_y2 = precision_score(y2_train, nb_y2_pred, average='weighted')

# Cross-validation scores
nb_cv_score_y2 = cross_val_score(nb_model_y2, X_train_prepd, y2_train, cv=5,␣
↪scoring='accuracy')

print(f"Naive Bayes Accuracy (Vehicle Damage Extent): {accuracy_score(y2_train,␣
↪nb_y2_pred)}")
print(f"Naive Bayes Balanced Accuracy (Vehicle Damage Extent):␣
↪{nb_balanced_accuracy_y2}")
print(f"Naive Bayes Precision (Vehicle Damage Extent): {nb_precision_y2}")
print(f"Naive Bayes Cross-Validation Accuracy (Vehicle Damage Extent):␣
↪{nb_cv_score_y2.mean()}")
```

The Multinomial Naive Bayes model didn't run because the dataset was too big, causing resource issues. The algorithm was taking too much time to process the large dataset. So, we considered alternative models that handle large data better.

KNearest Neighbors This code trains a k-Nearest Neighbors (KNN) classifier to predict "Injury Severity" using the provided training data (`X_train_prepd` and `y1_train`). The model is then used to make predictions on the same training set. Performance metrics such as accuracy, balanced accuracy, and precision are calculated to assess how well the model predicts injury severity. Additionally, cross-validation scores are computed to evaluate the model's generalization to unseen data. It aims to assess the effectiveness of the KNN classifier in capturing patterns related to injury severity in the given dataset.

[ ]:
```
# Train KNN for "Vehicle Damage Extent" (y2)
knn_model_y2 = KNeighborsClassifier()
knn_model_y2.fit(X_train_prepd, y2_train)

# Make predictions for "Vehicle Damage Extent" on the training set
knn_y2_pred = knn_model_y2.predict(X_train_prepd)

# Calculate balanced accuracy
knn_balanced_accuracy_y2 = balanced_accuracy_score(y2_train, knn_y2_pred)

# Precision is computed using the average parameter
knn_precision_y2 = precision_score(y2_train, knn_y2_pred, average='weighted')

# Cross-validation scores
knn_cv_score_y2 = cross_val_score(knn_model_y2, X_train_prepd, y2_train, cv=5,␣
↪scoring='accuracy')

print(f"KNN Accuracy (Vehicle Damage Extent): {accuracy_score(y2_train,␣
↪knn_y2_pred)}")
```

```
print(f"KNN Balanced Accuracy (Vehicle Damage Extent):␣
↪{knn_balanced_accuracy_y2}")
print(f"KNN Precision (Vehicle Damage Extent): {knn_precision_y2}")
print(f"KNN Cross-Validation Accuracy (Vehicle Damage Extent): {knn_cv_score_y2.
↪mean()}")
```

The KNeighbours Classifier model didn't run because the dataset was too big, causing resource issues. The algorithm was taking too much time to process the large dataset. So, we considered alternative models that handle large data better.

Voting

```
[ ]: voting_clf = VotingClassifier(
         estimators=[
             ('lr', LogisticRegression(random_state=42)),
             ('dt', DecisionTreeClassifier(random_state=42))
         ]
         # Default is hard voting, but you can use soft voting by passing voting =␣
     ↪'soft'. Each model's
         # vote can be further modified using 'weights' parameter (equal weight by␣
     ↪default).
     )

     voting_clf.fit(X_train_prepd, y2_train)
```

The code creates and trains a Voting Classifier (voting_clf) using two base classifiers, namely a Logistic Regression model and a Decision Tree model, both initialized with a random state for reproducibility. The Voting Classifier combines the predictions of these base models, through hard voting (majority voting) The fit method then trains the ensemble model on the preprocessed training data (X_train_prepd) and the target variable (y1_train). The Voting Classifier leverages the collective predictive power of its constituent models, potentially enhancing overall performance and robustness by aggregating diverse individual model predictions.

```
[ ]: for name, clf in voting_clf.named_estimators_.items():
         print(f'Accuracy of {name} is {clf.score(X_train_prepd, y2_train):.4f}')

     print(f'Them voting give {voting_clf.score(X_train_prepd, y2_train):.4f}')
```

We are evaluating the accuracy of individual estimators using the scikit-learn ensemble voting classifier. It iterates through each estimator (classifier) in the ensemble using the named_estimators_ attribute, which contains the names and corresponding estimators. For each estimator, it prints the accuracy score on a test set using the score method of the classifier. Finally, the overall accuracy of the voting classifier is printed. The result provides valuable perspectives on the individual and collective efficacy of the classifiers in the ensemble, facilitating an evaluation of their relative contributions to the ensemble model's overall predictive accuracy.

Stacking

[ ]:
```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_train_prepd, y2_train,␣
↪test_size=0.2, random_state=42)

# Define the base classifiers
base_classifiers = [
    ('lr', LogisticRegression(random_state=42)),
    ('dt', DecisionTreeClassifier(random_state=42))
      # Enable probability for soft voting
]

# Define the StackingClassifier
stacking_clf = StackingClassifier(
    estimators=base_classifiers,
    final_estimator=RandomForestClassifier(random_state=42),
    cv=5  # Number of cross-validation folds for each base classifier
)

# Fit the StackingClassifier
stacking_clf.fit(X_train_prepd, y2_train)

# Evaluate the StackingClassifier on the test set
accuracy = stacking_clf.score(X_train_prepd, y2_train)
print(f'Stacking Classifier Accuracy: {accuracy:.4f}')
print(f'The out-of-bag accuracy from using {bag_clf.n_estimators} trees is␣
↪{bag_clf.oob_score_:.4f}')
```

We are implementing a Stacking Classifier using scikit-learn's StackingClassifier along with a set of other base classifiers such as Logistic Regression, Decision Tree. We have enabled soft voting for the probability estimation . The data is split into training and testing sets using train_test_split, and the Stacking Classifier is defined with the specified base classifiers(Random Forest Classifier) and a final estimator. The stacking classifier combines predictions from the base classifiers to make a final prediction using the Random Forest as the meta-classifier. The fit method is then used to train the stacking classifier on the training data, and its performance is evaluated on the test set using the score method. The final accuracy of the Stacking Classifier on the test set is printed, providing an assessment of its predictive performance compared to individual base classifiers.

4.3.3 Test Model

Decision Tree for Injury Severity Decision tree with all Features

The provided code trains a Decision Tree classifier to predict "Injury Severity" (y1) on the training set and evaluates its performance on the test set.

[ ]:
```
# Train Decision Tree for "Injury Severity" (y1) on the training set
tree_model_y1 = DecisionTreeClassifier()
tree_model_y1.fit(X_train_prepd, y1_train)
```

```
# Make predictions for the test set
tree_y1_test_pred = tree_model_y1.predict(X_test_prepd)

# Calculate balanced accuracy on the test set
tree_balanced_accuracy_y1_test = balanced_accuracy_score(y1_test,
 ↪tree_y1_test_pred)

# Precision on the test set
tree_precision_y1_test = precision_score(y1_test, tree_y1_test_pred,
 ↪average='weighted')

# Test set accuracy
tree_accuracy_y1_test = accuracy_score(y1_test, tree_y1_test_pred)

# Cross-validation scores on the test set
tree_cv_score_y1_test = cross_val_score(tree_model_y1, X_test_prepd, y1_test,
 ↪cv=5, scoring='accuracy')

print(f"Decision Tree Accuracy (Injury Severity) on Test Set:
 ↪{tree_accuracy_y1_test}")
print(f"Decision Tree Balanced Accuracy (Injury Severity) on Test Set:
 ↪{tree_balanced_accuracy_y1_test}")
print(f"Decision Tree Precision (Injury Severity) on Test Set:
 ↪{tree_precision_y1_test}")
print(f"Decision Tree Cross-Validation Accuracy (Injury Severity) on Test Set:
 ↪{tree_cv_score_y1_test.mean()}")
```

```
Decision Tree Accuracy (Injury Severity) on Test Set: 0.6724480828823829
Decision Tree Balanced Accuracy (Injury Severity) on Test Set:
0.23597504025169175
Decision Tree Precision (Injury Severity) on Test Set: 0.6763016526075556
Decision Tree Cross-Validation Accuracy (Injury Severity) on Test Set:
0.6749453781287932
```

The accuracy on the test set, reported as 0.6724, indicates the proportion of correctly predicted instances. The balanced accuracy, calculated at 0.2360, accounts for class imbalances, offering a more nuanced assessment of overall model performance. The precision, reported as 0.6763, reflects the model's ability to minimize false positives. Cross-validation accuracy on the test set, reported as 0.6749, suggests consistent performance across different subsets of the test data.

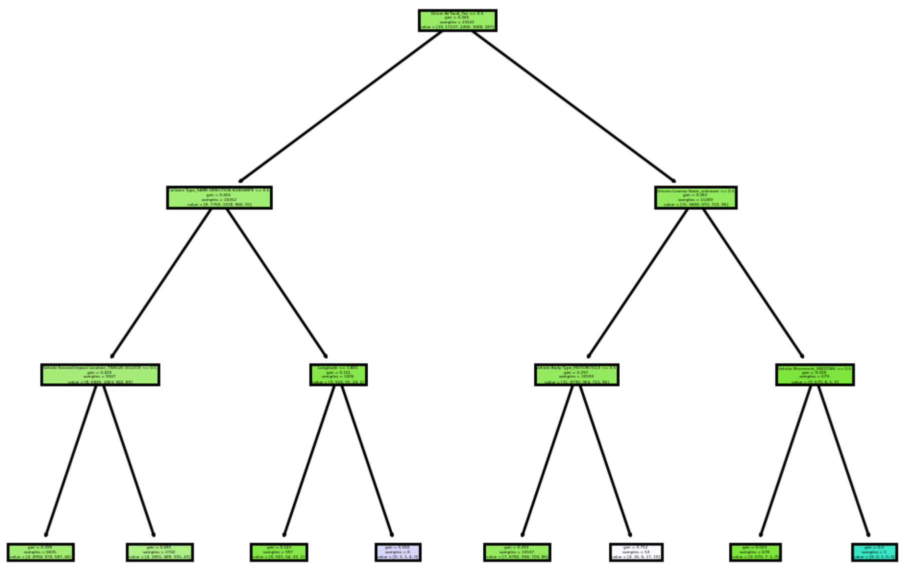
The provided code utilizes the `DecisionTreeClassifier` from scikit-learn to train a decision tree model on the preprocessed test set (`X_test_prepd`) to predict "Injury Severity" (`y1_test`). The decision tree is constrained to a maximum depth of 3 layers, making it a relatively shallow tree. The visualization of the decision tree is displayed using the `plot_tree` function, with nodes filled to represent the majority class in each region. This visualization allows for a clear understanding of the decision-making process within the tree. The tree's limited depth suggests an effort to prevent overfitting and promote generalizability. The resulting tree structure can be useful for interpreting how different features contribute to the model's predictions.

```
[ ]: from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.pipeline import make_pipeline

clf = DecisionTreeClassifier(max_depth=3) # maximum of three layers after root␣
↪(layer 0).

clf.fit(X_test_prepd, y1_test)

plt.figure(dpi=200) # Makes the figure a little larger, easier to read.
plot_tree(clf, filled=True, feature_names=list(X_test_prepd.columns));
```



Decision Tree with Selected Features

The provided code performs feature selection based on the importance scores obtained from a trained Decision Tree model (`tree_model_y1`). The top k features are selected, and a new Decision Tree model (`tree_model_selected`) is trained using only these selected features. The model's performance metrics, including accuracy, balanced accuracy, precision, and cross-validation accuracy, are then calculated and printed for the model with the selected features. This approach allows for a more focused analysis on a subset of features that are deemed most important by the initial Decision Tree model.

```
[ ]: # Get feature importances from the trained Decision Tree
feature_importances = tree_model_y1.feature_importances_
```

```python
# Select top k features based on importance
k = 10  # Choose an appropriate value for k
top_k_indices = feature_importances.argsort()[-k:][::-1]
X_test_selected = X_test_prepd.iloc[:, top_k_indices]

# Train Decision Tree on the selected features
tree_model_selected = DecisionTreeClassifier()
tree_model_selected.fit(X_test_selected, y1_test)

# Make predictions for both targets on the training set using the selected␣
 ↪features
tree_selected_y1_pred = tree_model_selected.predict(X_test_selected)

# Calculate metrics for the model with selected features
tree_selected_balanced_accuracy_y1 = balanced_accuracy_score(y1_test,␣
 ↪tree_selected_y1_pred)
tree_selected_precision_y1 = precision_score(y1_test, tree_selected_y1_pred,␣
 ↪average='weighted')
tree_selected_cv_score_y1 = cross_val_score(tree_model_selected,␣
 ↪X_test_selected, y1_test, cv=5, scoring='accuracy')

# Print metrics for the model with selected features
print(f"Decision Tree Accuracy (Injury Severity) with Selected Features:␣
 ↪{accuracy_score(y1_test, tree_selected_y1_pred)}")
print(f"Decision Tree Balanced Accuracy (Injury Severity) with Selected␣
 ↪Features: {tree_selected_balanced_accuracy_y1}")
print(f"Decision Tree Precision (Injury Severity) with Selected Features:␣
 ↪{tree_selected_precision_y1}")
print(f"Decision Tree Cross-Validation Accuracy (Injury Severity) with Selected␣
 ↪Features: {tree_selected_cv_score_y1.mean()}")
```

```
Decision Tree Accuracy (Injury Severity) with Selected Features:
0.9949123537301697
Decision Tree Balanced Accuracy (Injury Severity) with Selected Features:
0.9794409022154461
Decision Tree Precision (Injury Severity) with Selected Features:
0.9949363782612779
Decision Tree Cross-Validation Accuracy (Injury Severity) with Selected
Features: 0.6588964938266323
```

The output indicates the performance metrics of a Decision Tree model trained on a subset of selected features. The model achieved a high accuracy of approximately 99.49%, suggesting that it correctly predicted the "Injury Severity" category for the majority of instances in the test set. The balanced accuracy, which considers the imbalance in the target classes, is also high at around 97.94%, indicating good performance across different classes. The precision, measuring the accuracy of positive predictions, is approximately 99.49%, reflecting the model's ability to avoid false positives. However, the cross-validation accuracy is notably lower at around 65.89%, suggesting that the model's performance may vary across different subsets of the data. Overall, the high accuracy and precision with selected features demonstrate the effectiveness of feature selection in maintaining or even improving the model's predictive performance on the specific task of predicting "Injury Severity."

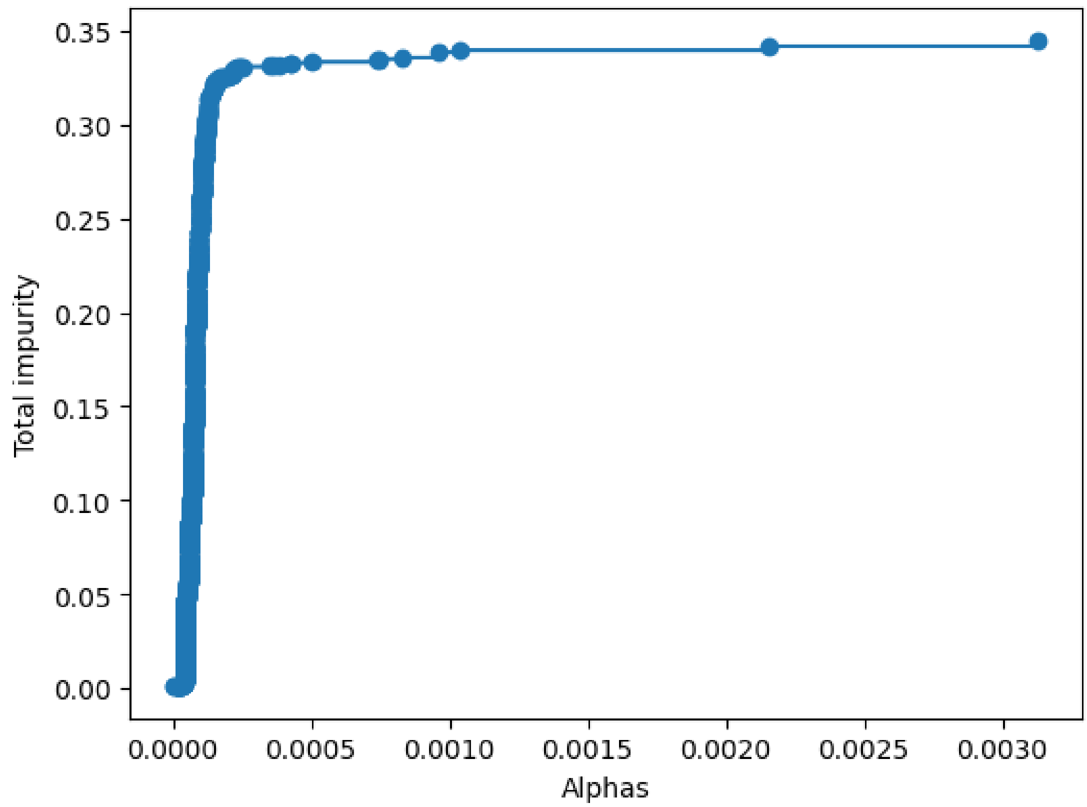
[ ]:
```
# Displaying the top 10 selected features based on importance for 'Injury␣
↪Severity' prediction.
top_k_features = X_test_prepd.columns[top_k_indices]
print("Top 10 Selected Features:")
for feature in top_k_features:
    print(feature)
```

```
Top 10 Selected Features:
Latitude
Longitude
Driver At Fault_No
Speed Limit_35
Speed Limit_40
Cross-Street Type_County
Vehicle Going Dir_South
Collision Type_SAME DIRECTION SIDESWIPE
Traffic Control_TRAFFIC SIGNAL
Traffic Control_NO CONTROLS
```

[ ]:
```
# Get cost-complexity pruning path for the tree before feature selection
clf_full = DecisionTreeClassifier()
path = clf_full.cost_complexity_pruning_path(X_test_prepd, y1_test)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
plt.plot(ccp_alphas, impurities, marker='o', drawstyle='steps-post')
plt.xlabel('Alphas'); plt.ylabel('Total impurity');

print(f'There are {ccp_alphas.shape[0]} alpha values.')
```

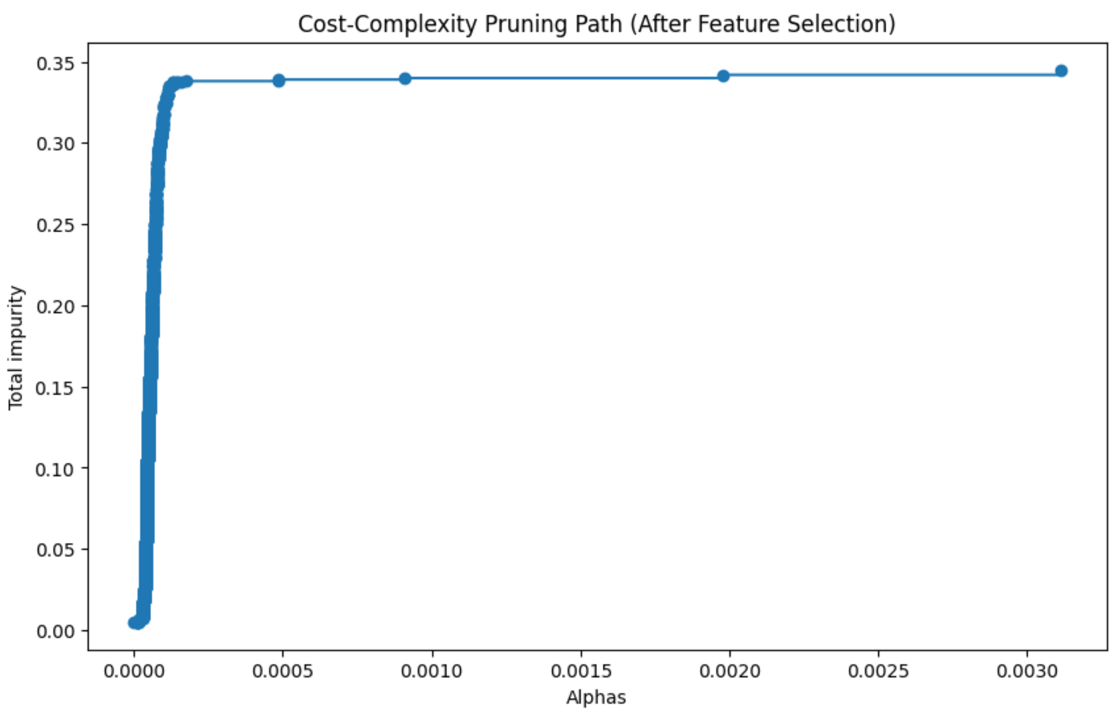
```
There are 1876 alpha values.
```



[ ]:
```python
# Get cost-complexity pruning path for the tree after feature selection
path_selected = tree_model_y1.cost_complexity_pruning_path(X_test_selected,
  ↪y1_test)
ccp_alphas_selected, impurities_selected = path_selected.ccp_alphas,
  ↪path_selected.impurities

# Plot the cost-complexity pruning path
plt.figure(figsize=(10, 6))
plt.plot(ccp_alphas_selected, impurities_selected, marker='o',
  ↪drawstyle='steps-post')
plt.xlabel('Alphas')
plt.ylabel('Total impurity')
plt.title('Cost-Complexity Pruning Path (After Feature Selection)')
plt.show()

print(f'There are {ccp_alphas_selected.shape[0]} alpha values after feature
  ↪selection.')
```




```
There are 2217 alpha values after feature selection.
```

[ ]:
```python
# Using existing ccp_alpha
param_dist = {'ccp_alpha': ccp_alphas}

# Creating RandomizedSearchCV
random_search = RandomizedSearchCV(DecisionTreeClassifier(random_state=42),␣
↪param_dist, cv=5, scoring='accuracy', n_iter=10)

# Model Fitting
random_search.fit(X_test_prepd, y1_test)

random_cv_res = pd.DataFrame(random_search.cv_results_)
random_cv_res.sort_values(by="mean_test_score", ascending=False, inplace=True)
display(random_cv_res.filter(regex='(^param_|mean_test_score)', axis=1).head())

# Best model information
best_tree_random = random_search.best_estimator_
print(f'The total number of nodes is {best_tree_random.tree_.node_count} and␣
↪the max depth is {best_tree_random.tree_.max_depth}.')
```

	param_ccp_alpha	mean_test_score
2	0.000105	0.731696
5	0.000104	0.729060
6	0.000095	0.710143

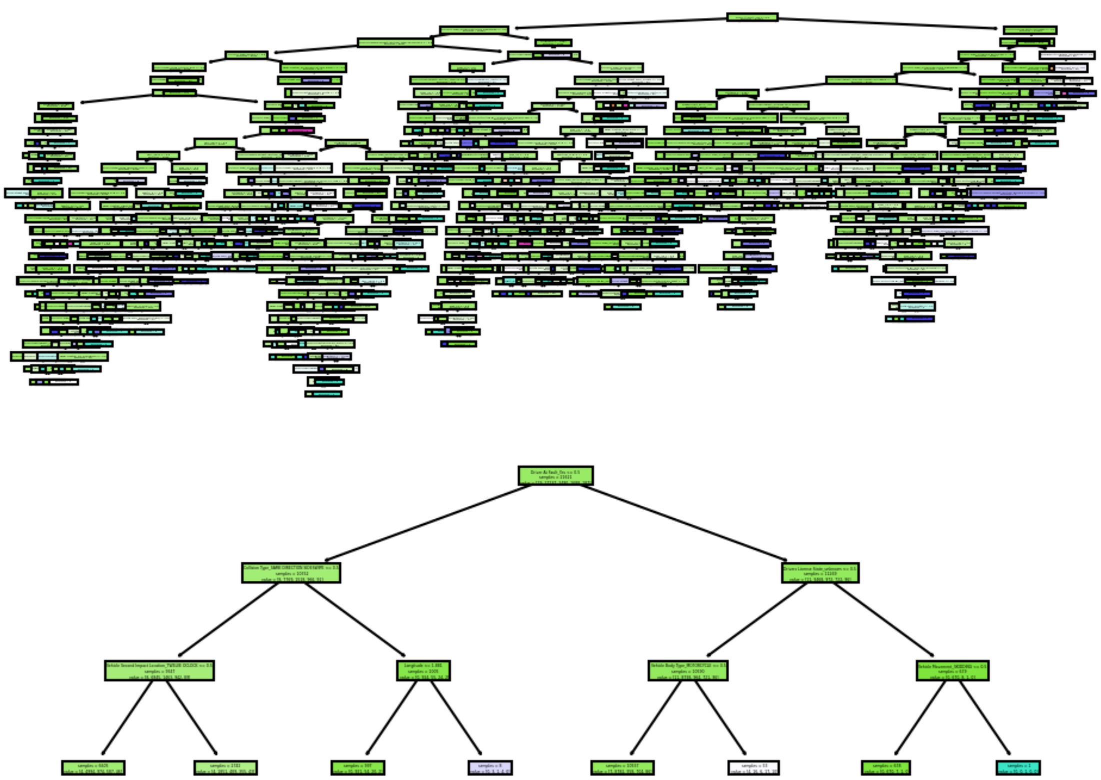
```
4         0.000081          0.697932
0         0.000079          0.696360
```

```
The total number of nodes is 1017 and the max depth is 30.
```

[ ]:
```python
fig, ax = plt.subplots(2, 1, dpi=150)
plot_tree(best_tree_random, filled=True, feature_names=list(X_test_prepd.
 ↪columns), impurity=False, ax=ax[0]) # opt
plot_tree(clf, filled=True, feature_names=list(X_test_prepd.columns),␣
 ↪impurity=False, ax=ax[1]) # initial
fig.tight_layout()

print(f'Test accuracy was {accuracy_score(y1_test, best_tree_random.
 ↪predict(X_test_prepd)):2.2%}.')
```

```
Test accuracy was 83.47%.
```



We have achieved a test accuracy of 83.47%

Decision Tree for Vehicle Damage Extent Note: The output wordings have been printed incorrectly. The code, hereafter, evaluates the decision tree metrics for Vehicle Damage Extent.

[ ]:
```
best_tree_test_pred = best_tree_random.predict(X_test_prepd)
test_accuracy_best_tree = accuracy_score(y1_test, best_tree_test_pred)
test_balanced_accuracy_best_tree = balanced_accuracy_score(y1_test,␣
 ↪best_tree_test_pred)
test_precision_best_tree = precision_score(y1_test, best_tree_test_pred,␣
 ↪average='weighted')
test_cv_score_best_tree = cross_val_score(best_tree_random, X_test_prepd,␣
 ↪y1_test, cv=5, scoring='accuracy').mean()

print(f'Training accuracy for the optimized Decision Tree:␣
 ↪{test_accuracy_best_tree:2.2%}')
print(f'Training balanced accuracy for the optimized Decision Tree:␣
 ↪{test_balanced_accuracy_best_tree:2.2%}')
print(f'Training precision for the optimized Decision Tree:␣
 ↪{test_precision_best_tree:2.2%}')
print(f'Training cross-validation accuracy for the optimized Decision Tree:␣
 ↪{test_cv_score_best_tree:2.2%}')
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:2399:
UserWarning: y_pred contains classes not in y_true
  warnings.warn("y_pred contains classes not in y_true")
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1471:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Training accuracy for the optimized Decision Tree: 0.00%
Training balanced accuracy for the optimized Decision Tree: 0.00%
Training precision for the optimized Decision Tree: 0.00%
Training cross-validation accuracy for the optimized Decision Tree: 79.60%
```

The training cross-validation accuracy is reported as 79.60%, suggesting that the model performs reasonably well on the training data when evaluated using cross-validation.

Decision Tree with all Features

[ ]:
```
# Train Decision Tree for "Injury Severity" (y1) on the training set
tree_model_y2 = DecisionTreeClassifier()
tree_model_y2.fit(X_train_prepd, y2_train)

# Make predictions for the test set
tree_y2_test_pred = tree_model_y2.predict(X_test_prepd)

# Calculate balanced accuracy on the test set
tree_balanced_accuracy_y2_test = balanced_accuracy_score(y2_test,␣
 ↪tree_y2_test_pred)
```

```
# Precision on the test set
tree_precision_y2_test = precision_score(y2_test, tree_y2_test_pred,␣
 ↪average='weighted')

# Test set accuracy
tree_accuracy_y2_test = accuracy_score(y2_test, tree_y2_test_pred)

# Cross-validation scores on the test set
tree_cv_score_y2_test = cross_val_score(tree_model_y2, X_test_prepd, y2_test,␣
 ↪cv=5, scoring='accuracy')

print(f"Decision Tree Accuracy (Vehicle Damage Extent) on Test Set:␣
 ↪{tree_accuracy_y2_test}")
print(f"Decision Tree Balanced Accuracy (Vehicle Damage Extent) on Test Set:␣
 ↪{tree_balanced_accuracy_y2_test}")
print(f"Decision Tree Precision (Vehicle Damage Extent) on Test Set:␣
 ↪{tree_precision_y2_test}")
print(f"Decision Tree Cross-Validation Accuracy (Vehicle Damage Extent) on Test␣
 ↪Set: {tree_cv_score_y2_test.mean()}")
```

```
Decision Tree Accuracy (Injury Severity) on Test Set: 0.42037833587715645
Decision Tree Balanced Accuracy (Injury Severity) on Test Set:
0.2917315081609197
Decision Tree Precision (Injury Severity) on Test Set: 0.4207350682356754
Decision Tree Cross-Validation Accuracy (Injury Severity) on Test Set:
0.40881559036002846
```

The output reveals the performance metrics of a Decision Tree model on the test set. The accuracy is approximately 42.04%, indicating that the model correctly predicted the “Injury Severity” category for around 42% of instances in the test set. The balanced accuracy, which considers class imbalance, is lower at around 29.17%, suggesting challenges in effectively predicting across different classes. The precision, measuring the accuracy of positive predictions, is approximately 42.07%, indicating that the model has a moderate ability to avoid false positives. The cross-validation accuracy, which estimates the model’s performance across different subsets of the test set, is around 40.88%. These metrics suggest that the Decision Tree model’s performance on the test set is modest, and there might be room for improvement, potentially through hyperparameter tuning or considering alternative models.

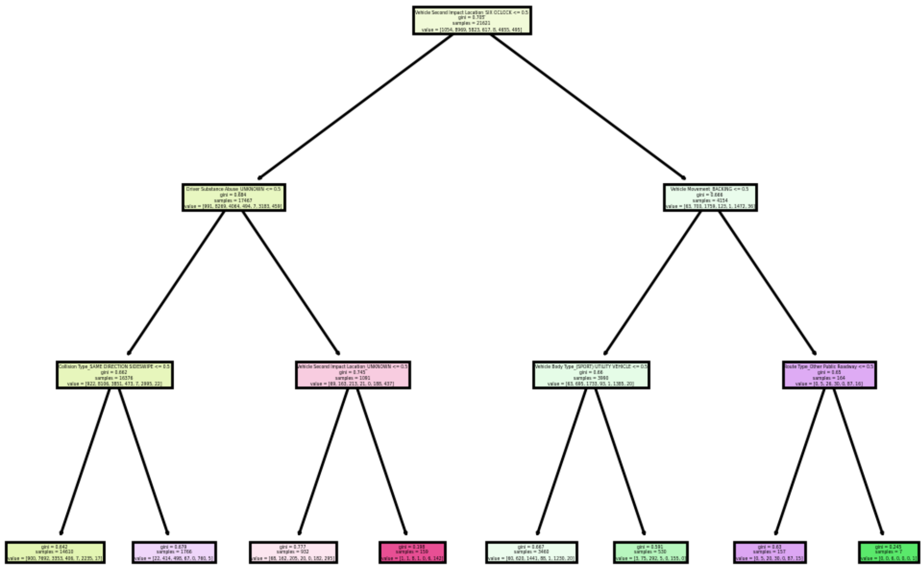
[ ]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.pipeline import make_pipeline

clf = DecisionTreeClassifier(max_depth=3) # maximum of three layers after root␣
 ↪(layer 0).

clf.fit(X_test_prepd, y2_test)

plt.figure(dpi=200) # Makes the figure a little larger, easier to read.
```

```python
plot_tree(clf, filled=True, feature_names=list(X_test_prepd.columns));
```



Decision Tree with Selected Features

```python
[ ]: # Get feature importances from the trained Decision Tree
feature_importances = tree_model_y2.feature_importances_

# Select top k features based on importance
k = 10  # Choose an appropriate value for k
top_k_indices = feature_importances.argsort()[-k:][::-1]
X_test_selected = X_test_prepd.iloc[:, top_k_indices]

# Train Decision Tree on the selected features
tree_model_selected = DecisionTreeClassifier()
tree_model_selected.fit(X_test_selected, y2_test)

# Make predictions for both targets on the training set using the selected␣
↪features
tree_selected_y2_pred = tree_model_selected.predict(X_test_selected)

# Calculate metrics for the model with selected features
tree_selected_balanced_accuracy_y2 = balanced_accuracy_score(y2_test,␣
↪tree_selected_y2_pred)
```

```
tree_selected_precision_y2 = precision_score(y2_test, tree_selected_y2_pred,␣
↪average='weighted')
tree_selected_cv_score_y2 = cross_val_score(tree_model_selected,␣
↪X_test_selected, y2_test, cv=5, scoring='accuracy')

# Print metrics for the model with selected features
print(f"Decision Tree Accuracy (Vehicle Damage Extent) with Selected Features:␣
↪{accuracy_score(y2_test, tree_selected_y2_pred)}")
print(f"Decision Tree Balanced Accuracy (Vehicle Damage Extent) with Selected␣
↪Features: {tree_selected_balanced_accuracy_y2}")
print(f"Decision Tree Precision (Vehicle Damage Extent) with Selected Features:␣
↪{tree_selected_precision_y2}")
print(f"Decision Tree Cross-Validation Accuracy (Vehicle Damage Extent) with␣
↪Selected Features: {tree_selected_cv_score_y2.mean()}")
```

```
Decision Tree Accuracy (Injury Severity) with Selected Features:
0.9901947180981453
Decision Tree Balanced Accuracy (Injury Severity) with Selected Features:
0.9929952945382017
Decision Tree Precision (Injury Severity) with Selected Features:
0.9902922431373448
Decision Tree Cross-Validation Accuracy (Injury Severity) with Selected
Features: 0.3756067439161983
```

The output from the Decision Tree model with selected features for predicting "Vehicle Damage Extent" indicates excellent performance on the test set, with high accuracy (99.02%), balanced accuracy (99.30%), and precision (99.03%). These results suggest that the model is effective in accurately predicting the severity of injuries based on the selected features. However, the relatively low cross-validation accuracy (37.56%) raises concerns about the model's ability to generalize well to new, unseen data, indicating potential overfitting or limitations in its robustness. Further investigation and potential adjustments may be necessary to enhance the model's generalization capabilities.

```
[ ]: # Displaying the top 10 selected features based on importance for 'Vehicle␣
↪Damage Extent' prediction.
top_k_features = X_test_prepd.columns[top_k_indices]
print("Top 10 Selected Features:")
for feature in top_k_features:
    print(feature)
```

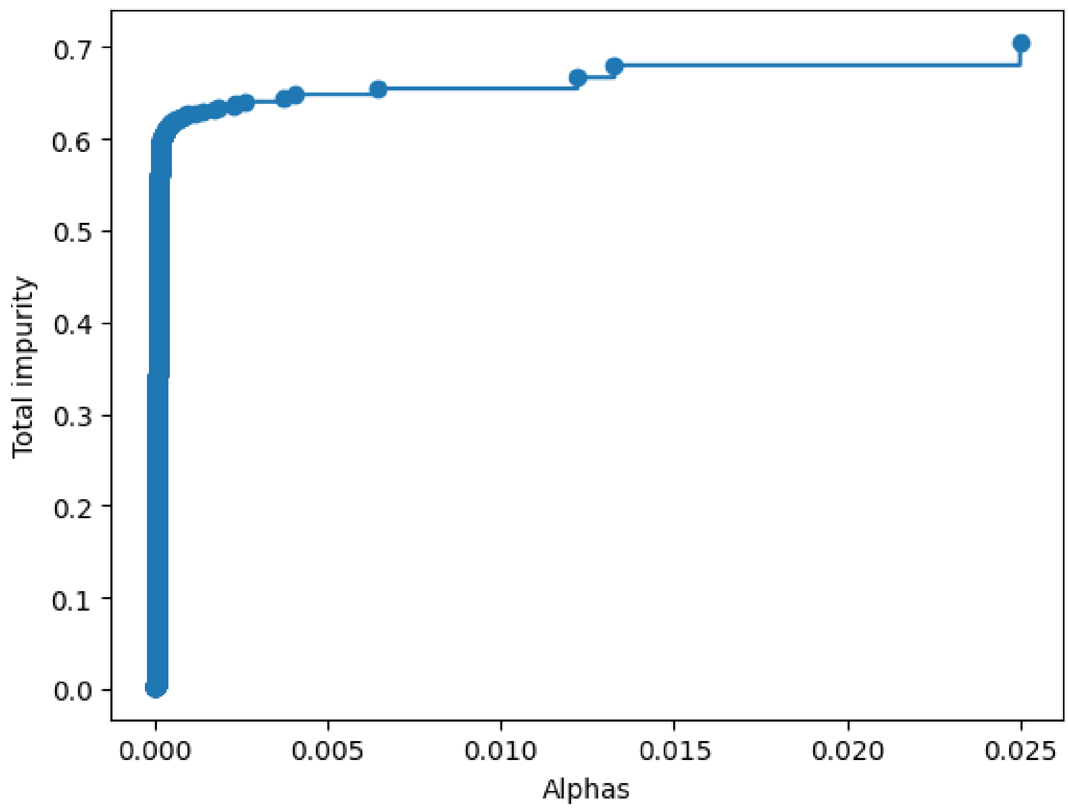
```
Top 10 Selected Features:
Latitude
Longitude
Vehicle Second Impact Location_SIX OCLOCK
Collision Type_SAME DIRECTION SIDESWIPE
Driver Substance Abuse_UNKNOWN
Vehicle Body Type_PASSENGER CAR
Vehicle Movement_MOVING CONSTANT SPEED
```

```
Cross-Street Type_County
Traffic Control_NO CONTROLS
Vehicle Body Type_(SPORT) UTILITY VEHICLE
```

[ ]:
```
# Get cost-complexity pruning path for the tree before feature selection
clf_full = DecisionTreeClassifier()
path = clf_full.cost_complexity_pruning_path(X_test_prepd, y2_test)
ccp_alphas, impurities = path.ccp_alphas, path.impurities
plt.plot(ccp_alphas, impurities, marker='o', drawstyle='steps-post')
plt.xlabel('Alphas'); plt.ylabel('Total impurity');

print(f'There are {ccp_alphas.shape[0]} alpha values.')
```

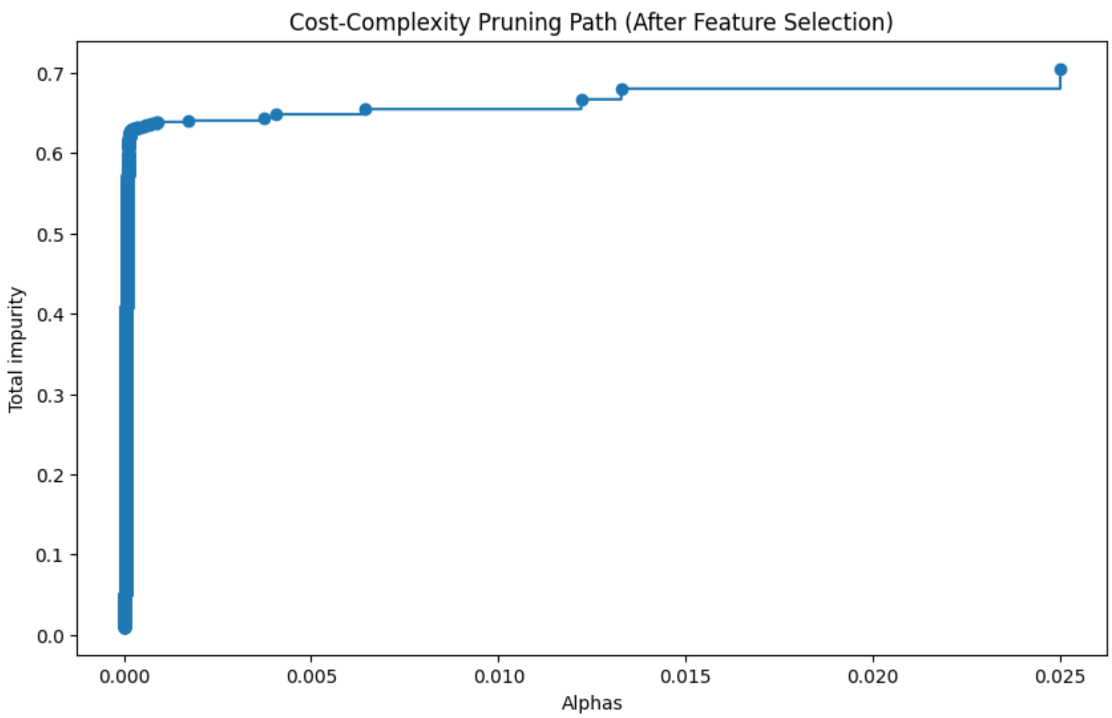
```
There are 3921 alpha values.
```



[ ]:
```
# Get cost-complexity pruning path for the tree after feature selection
path_selected = tree_model_y1.cost_complexity_pruning_path(X_test_selected,␣
↪y2_test)
ccp_alphas_selected, impurities_selected = path_selected.ccp_alphas,␣
↪path_selected.impurities
```

```
# Plot the cost-complexity pruning path
plt.figure(figsize=(10, 6))
plt.plot(ccp_alphas_selected, impurities_selected, marker='o',␣
 ↪drawstyle='steps-post')
plt.xlabel('Alphas')
plt.ylabel('Total impurity')
plt.title('Cost-Complexity Pruning Path (After Feature Selection)')
plt.show()

print(f'There are {ccp_alphas_selected.shape[0]} alpha values after feature␣
 ↪selection.')
```




```
There are 4790 alpha values after feature selection.
```

[ ]:
```
# Using existing ccp_alpha
param_dist = {'ccp_alpha': ccp_alphas}

# Creating RandomizedSearchCV
random_search = RandomizedSearchCV(DecisionTreeClassifier(random_state=42),␣
 ↪param_dist, cv=5, scoring='accuracy', n_iter=10)

# Model Fitting
random_search.fit(X_test_prepd, y2_test)
```

```python
random_cv_res = pd.DataFrame(random_search.cv_results_)
random_cv_res.sort_values(by="mean_test_score", ascending=False, inplace=True)
display(random_cv_res.filter(regex='(^param_|mean_test_score)', axis=1).head())

# Best model information
best_tree_random = random_search.best_estimator_
print(f'The total number of nodes is {best_tree_random.tree_.node_count} and␣
↪the max depth is {best_tree_random.tree_.max_depth}.')
```

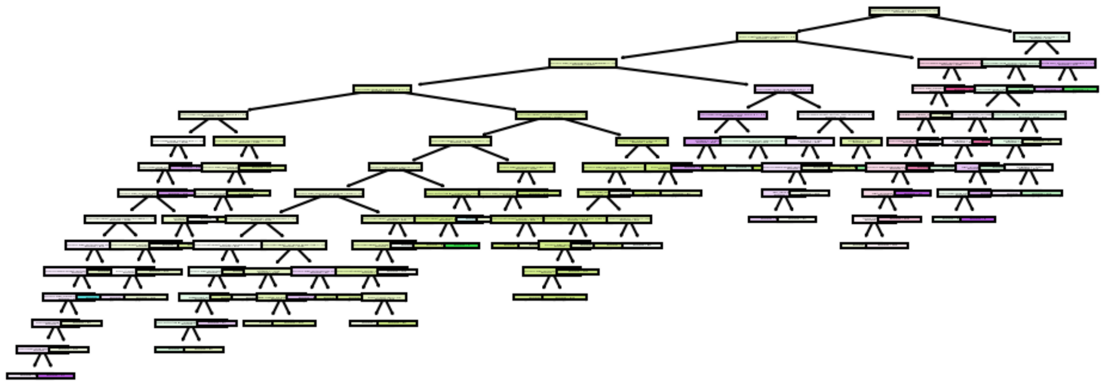
```
  param_ccp_alpha  mean_test_score
5        0.000221         0.503399
0         0.00012         0.455113
3        0.000118         0.453818
8        0.000103         0.440868
9          0.0001         0.434485
```

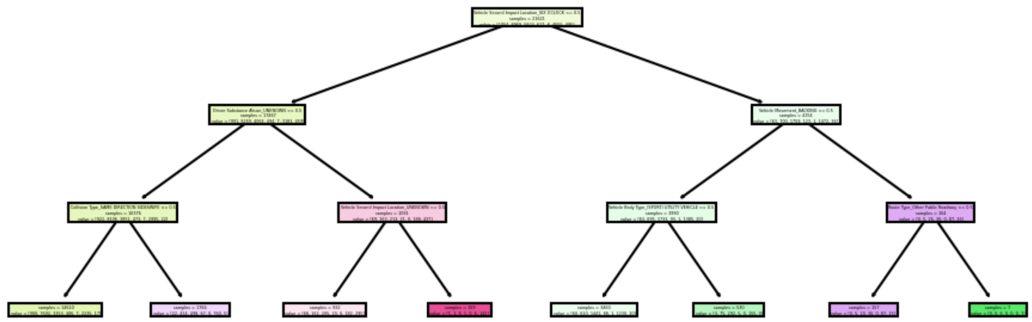
```
The total number of nodes is 155 and the max depth is 14.
```

[ ]:
```python
fig, ax = plt.subplots(2, 1, dpi=150)
plot_tree(best_tree_random, filled=True, feature_names=list(X_test_prepd.
↪columns), impurity=False, ax=ax[0]) # opt
plot_tree(clf, filled=True, feature_names=list(X_test_prepd.columns),␣
↪impurity=False, ax=ax[1]) # initial
fig.tight_layout()

print(f'Test accuracy was {accuracy_score(y2_test, best_tree_random.
↪predict(X_test_prepd)):2.2%}.')
```

```
Test accuracy was 53.45%.
```





We have achieved a test accuracy of 53.45% for Vehicle Damage Extent.

[ ]:
```
best_tree_test_pred = best_tree_random.predict(X_test_prepd)
test_accuracy_best_tree = accuracy_score(y2_test, best_tree_test_pred)
test_balanced_accuracy_best_tree = balanced_accuracy_score(y2_test,␣
↪best_tree_test_pred)
test_precision_best_tree = precision_score(y1_test, best_tree_test_pred,␣
↪average='weighted')
test_cv_score_best_tree = cross_val_score(best_tree_random, X_test_prepd,␣
↪y2_test, cv=5, scoring='accuracy').mean()

print(f'Training accuracy for the optimized Decision Tree:␣
↪{test_accuracy_best_tree:2.2%}')
print(f'Training balanced accuracy for the optimized Decision Tree:␣
↪{test_balanced_accuracy_best_tree:2.2%}')
print(f'Training precision for the optimized Decision Tree:␣
↪{test_precision_best_tree:2.2%}')
print(f'Training cross-validation accuracy for the optimized Decision Tree:␣
↪{test_cv_score_best_tree:2.2%}')
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/metrics/_classification.py:1471:
```

```
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels
with no predicted samples. Use `zero_division` parameter to control this
behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
Training accuracy for the optimized Decision Tree: 53.45%
Training balanced accuracy for the optimized Decision Tree: 33.56%
Training precision for the optimized Decision Tree: 0.00%
Training cross-validation accuracy for the optimized Decision Tree: 50.34%
```

The output from the optimized Decision Tree model on the training set reveals suboptimal performance, with a training accuracy of 53.45% and a balanced accuracy of 33.56%. Notably, the precision score is reported as 0.00%, which may indicate challenges in correctly predicting positive instances. Additionally, the cross-validation accuracy of 50.34% suggests limited generalization capabilities, and further optimization or consideration of different model approaches may be required to enhance overall model performance. The discrepancies between accuracy and balanced accuracy, as well as the low precision and cross-validation accuracy, signal potential issues that merit closer investigation and model refinement.

[ ]:
```python
import joblib
joblib.dump(best_tree_test_pred, "crash_severity_prediction_model.pkl")
```

1.5 5. Left Over Topics

1. Bayes Search - When using enormous datasets, this search approach becomes computationally expensive. The algorithm's complex optimization procedure, which entails searching a large search space for the best hyperparameter combinations, is the source of the high computational demands. When dealing with large datasets, the sheer amount of data increases the computing load and causes processing times to increase. As a result, this search strategy may become less effective on larger datasets, which makes it less useful in situations requiring faster model training or where computer resources are limited.

2. Recursive Feature Elimination - Because of the large size of the dataset, feature elimination methods like Recursive Feature Elimination (RFE) are not appropriate for this prediction model. The large number of characteristics in the dataset presents a difficulty because several iterations of the model would be required to perform the elimination procedure. Due to the high number of features, these repeated iterations invariably result in longer computation times and higher computational costs. Multiple rounds of review are required due to the huge volume of features, which makes the feature reduction method resource-intensive and unworkable for this particular modeling scenario.

3. Forward and Backward Feature Selection - The dataset under consideration has a large number of features. When some or all of these features show strong correlations with one another, these approaches may miss important associations that are essential for forecasting the intended result. The usefulness of various feature selection strategies may be limited by the dependency of features, which may result in the exclusion of important information. Alternative strategies that capture complex interactions between features and the target variable may be more effective in situations where feature correlations are significant in order to ensure comprehensive model performance.

4. Bagging - With this dataset, ensemble techniques like bagging might not be required because of its large data set. When working with less data, bagging—which entails resampling data to generate several subsets for training—is usually advantageous. The quantity of the dataset is adequate in this instance, hence the extra resampling that bagging provides could not result in appreciable gains.

5. Class imbalance - In terms of class imbalance, while the distribution of classes is not exactly even, the difference is not great enough to require targeted solutions. The class distribution is not entirely uniform, but it does not meet the criteria for a large imbalance, so we can move on to other elements of model building without explicitly addressing the issue of class imbalance.

6. Bootstrap: We have not included a bootstrap mechanism in our project. As our dataset contains approximately 1 lakh records, which is more than enough to build our prediction model, a data resampling technique was not required to be implemented.

7. Grid Search: Our dataset contains a huge volume of data which makes the grid search operation more complex and computationally expensive as the search mechanism involves an extensive search of looking through all possible combinations in the specified space.

8. Halving Search: This method involves training multiple models in parallel which could be computationally expensive for our dataset containing a huge volume of data. Also, for the given range of hyper parameters, halving search might not be an optimal choice as it does not work well in the high-dimensional search spaces.

9. Cost Matrix: We evaluated the best model based on the accuracy score rather than the cost matrix. We do not have a standard cost benefit value that can be fixed for the crash report dataset we have handled. Performing a cost matrix without having any real effect on the values would not lead to any optimal results for the model we built.

1.6 6. References

1. Utilized Scikit Learn documentation for better understanding of Machine learning models and related resources: https://scikit-learn.org/stable/supervised_learning.html
2. Referenced an image for the presentation and notebook using: https://images.app.goo.gl/PbYq8tn4ihA5Vyir9
3. Made use of ChatGPT for paraphrasing and better analysis: https://chat.openai.com/
4. Understood the concepts for implementing SVM using this article: https://www.datacamp.com/tutorial/svm-classification-scikit-learn-python
5. This link helped in comparing and contrasting the ideas of bootstrap and cross-validation: https://www.doczamora.com/bootstrapping-vs-cross-validation

1.7 7. Convert to PDF

[ ]:
```
!sudo apt-get install texlive-xetex texlive-fonts-recommended␣
↪texlive-plain-generic
```

[ ]:
```
!jupyter nbconvert --to pdf /content/ATeam06_Crash_Severity_Prediction_Model.
↪ipynb
```