National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Joseph Agoi
Student ID:x22121684

School of Computing
National College of Ireland

Supervisor: Supervisor: Dr. Anu Sahni

National College of Ireland
Project Submission Sheet
School of Computing

National College of Ireland

Student Name:	Joseph Agoi
Student ID:	x22121684
Programme:	Data Analytics
Year:	2023
Module:	MSc Research Project
Supervisor:	Dr. Anu Sahni
Submission Due Date:	14/12/2023
Project Title:	Configuration Manual
Word Count:	1,165
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Joseph Agoi
Date:	31st January 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Joseph Agoi
x22121684

1 Introduction

This document highlights the necessary steps and instructions to replicate the study on a predictive analysis project for Chicago crash severity and obtain the expected results. It captures the system configurations along with the project development process. The code snippets for the implementation process will be included.

2 System Configuration

Due to the sheer volume of data used for the study, a higher hardware specification is required to effectively carry out the study. Figure 1 below captures the hardware specification used to achieve the research objectives. The software requirements are captured in Figure 2.

Hardware Configurations	
Item	Details
Operating System	Windows 10 Pro
Installed RAM	24 GB
Hard Disk Space	256 GB
Processor	Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz 2.90 GHz

Figure 1: Hardware Specifications

Software Used to Deliver the Project	
Item	Details
Programming Language	Python 3.7.2
Integrated Development Environment (IDE)	Jupyter Notebook 5.7.6
Other Software Utilized	Microsoft Office Suite

Figure 2: Software Requirements

3 Project Development

The following steps provide a view of the code sequences executed within Jupyter Notebook, allowing you to understand the specific code snippets, commands, functions, or algorithms, and the outcomes produced during the execution.

3.1 Data Collection

As shown in Figure 3, an API call was made to the city of Chicago's Data portal API endpoint. The data was saved in a local drive in an Excel format.

```python
import requests
import csv

url = 'https://data.cityofchicago.org/resource/85ca-t3if.json'

response = requests.get(url)

if response.status_code == 200:
    data = response.json()

    # file path to save the CSV file
    file_path = 'C:\Chicago Traffic Crashes.csv'

    # Write data to a CSV file
    with open(file_path, 'w', newline='') as file:
        csv_writer = csv.writer(file)

        # Write the header
        header = list(data[0].keys())
        csv_writer.writerow(header)

        # Write each row of data to the CSV file
        for item in data:
            csv_writer.writerow(item.values())

    print(f"CSV file downloaded successfully to: {file_path}")
else:
    print("Failed to fetch data from the API")
```

Figure 3: API Code Snippet

Upon importing the retrieved dataset, a preliminary assessment was conducted to ensure its integrity. Following this, a filtering process was implemented to exclusively retain records corresponding to the timeframe encompassing the years 2021 to 2022.

3.2 Data Importation

The data was imported into a Python data frame as shown in figure 4. Basic info was printed out to have an understanding of the dataset.

```python
import pandas as pd

df = pd.read_excel('Chicago Traffic Crashes.xlsx')

# Display basic information about the dataset
print(df.info())
```

Figure 4: API Code Snippet

3.3 Data Pre-processing

Variables that were not core to the study were identified and dropped as shown in Figure 5 below.

```
columns_to_drop = ['LANE_CNT','WORKERS_PRESENT_I','DOORING_I','WORK_ZONE_TYPE','WORK_ZONE_I','PHOTOS_TAKEN_I',
                   'STATEMENTS_TAKEN_I','NOT_RIGHT_OF_WAY_I','CRASH_DATE_EST_I','INTERSECTION_RELATED_I','HIT_AND_RUN_I',
                   'REPORT_TYPE','LOCATION','INJURIES_INCAPACITATING','INJURIES_NON_INCAPACITATING',
                   'INJURIES_REPORTED_NOT_EVIDENT','INJURIES_UNKNOWN','INJURIES_FATAL','STREET_DIRECTION',
                   'CRASH_RECORD_ID','BEAT_OF_OCCURRENCE','STREET_NAME','CRASH_DATE','DATE_POLICE_NOTIFIED',
                   'SEC_CONTRIBUTORY_CAUSE','RD_NO','STREET_NO']

new_df = df.drop(columns=columns_to_drop)
```

Figure 5: Dropped Variables

Label encoding and manual mapping were done on various categorical variables in the new data frame.

Figures 6, 7, 8, and 9 detail label encoding and mapping procedures for selected categorical variables in a dataset. They provide step-by-step insights into the conversion and mapping of these variables, enabling numerical analysis and computation.

```
trafficway_map = {
    'ONE-WAY': 1,
    'UNKNOWN': 2,
    'NOT DIVIDED': 3,
    'DIVIDED - W/MEDIAN BARRIER': 4,
    'FOUR WAY': 5,
    'DIVIDED - W/MEDIAN (NOT RAISED)': 6,
    'T-INTERSECTION': 7,
    'RAMP': 8,
    'CENTER TURN LANE': 9,
    'ALLEY': 10,
    'FIVE POINT, OR MORE': 11,
    'Y-INTERSECTION': 12,
    'PARKING LOT': 13,
    'UNKNOWN INTERSECTION TYPE': 14,
    'OTHER': 15,
    'DRIVEWAY': 16,
    'TRAFFIC ROUTE': 17,
    'NOT REPORTED': 18,
    'ROUNDABOUT': 19,
    'L-INTERSECTION': 20
}

# Map the 'TRAFFICWAY_TYPE' column using the dictionary
new_df['TRAFFICWAY_TYPE_NUMERIC'] = new_df['TRAFFICWAY_TYPE'].map(trafficway_map)
new_df.head()
```

Figure 6: Trafficway_Type label encoding and mapping

Label encoding and manual mapping were done on various categorical variables in the new data frame.

```
alignment_mapping = {
    'STRAIGHT AND LEVEL': 1,
    'STRAIGHT ON GRADE': 2,
    'CURVE, LEVEL': 3,
    'STRAIGHT ON HILLCREST': 4,
    'CURVE ON GRADE': 5,
    'CURVE ON HILLCREST': 6
}

# Replace 'ALIGNMENT' values with their corresponding numeric representations
new_df['ALIGNMENT_NUMERIC'] = new_df['ALIGNMENT'].map(alignment_mapping)

new_df.head()
```

Figure 7: Road_Alignment label encoding and mapping

```
road_defect_mapping = {
    'NO DEFECTS': 1,
    'UNKNOWN': 2,
    'RUT, HOLES': 3,
    'WORN SURFACE': 4,
    'OTHER': 5,
    'SHOULDER DEFECT': 6,
    'DEBRIS ON ROADWAY': 7
}

# Replace 'ROAD_DEFECT' values with their corresponding numeric representations
new_df['ROAD_DEFECT_NUMERIC'] = new_df['ROAD_DEFECT'].map(road_defect_mapping)
new_df.head()
```

Figure 8: Road_Defect label encoding and mapping

```
weather_condition_mapping = {
    'CLEAR': 1,
    'UNKNOWN': 2,
    'RAIN': 3,
    'FOG/SMOKE/HAZE': 4,
    'OTHER': 5,
    'CLOUDY/OVERCAST': 6,
    'SNOW': 7,
    'FREEZING RAIN/DRIZZLE': 8,
    'SLEET/HAIL': 9,
    'BLOWING SNOW': 10,
    'SEVERE CROSS WIND GATE': 11,
    'BLOWING SAND, SOIL, DIRT': 12
}

# Replace 'WEATHER_CONDITION' values with their corresponding numeric representations
new_df['WEATHER_CONDITION_NUMERIC'] = new_df['WEATHER_CONDITION'].map(weather_condition_mapping)
new_df.head()
```

Figure 9: Weather_Condition label encoding and mapping

As illustrated in Figure 10, the original categorical columns were removed from the dataset after label encoding and mapping, allowing for streamlined analysis and computation, eliminating redundancy.

```
new_df.columns
```

```
Index(['POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION',
       'WEATHER_CONDITION', 'LIGHTING_CONDITION', 'FIRST_CRASH_TYPE',
       'TRAFFICWAY_TYPE', 'ALIGNMENT', 'ROADWAY_SURFACE_COND', 'ROAD_DEFECT',
       'CRASH_TYPE', 'DAMAGE', 'PRIM_CONTRIBUTORY_CAUSE', 'NUM_UNITS',
       'MOST_SEVERE_INJURY', 'INJURIES_TOTAL', 'INJURIES_NO_INDICATION',
       'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH', 'LATITUDE',
       'LONGITUDE', 'SEVERE_INJURY', 'POSTED_SPEED_LIMIT_CATEGORY',
       'DAY_NIGHT_INDICATOR', 'TRAFFICWAY_TYPE_NUMERIC', 'ALIGNMENT_NUMERIC',
       'ROADWAY_SURFACE_NUMERIC', 'ROAD_DEFECT_NUMERIC', 'CRASH_TYPE_NUMERIC',
       'TRAFFIC_CONTROL_DEVICE_NUMERIC', 'DEVICE_CONDITION_NUMERIC',
       'WEATHER_CONDITION_NUMERIC', 'FIRST_CRASH_TYPE_NUMERIC',
       'DAMAGE_NUMERIC', 'PRIM_CONTRIBUTORY_CAUSE_NUMERIC'],
      dtype='object')
```

```
columns_to_drop = ['TRAFFIC_CONTROL_DEVICE', 'DEVICE_CONDITION', 'WEATHER_CONDITION', 'LIGHTING_CONDITION',
                   'FIRST_CRASH_TYPE', 'TRAFFICWAY_TYPE', 'ALIGNMENT', 'ROADWAY_SURFACE_COND', 'ROAD_DEFECT',
                   'CRASH_TYPE', 'DAMAGE', 'PRIM_CONTRIBUTORY_CAUSE','MOST_SEVERE_INJURY']

# Drop the specified columns from the DataFrame
new_df = new_df.drop(columns=columns_to_drop, axis=1)
new_df.head()
```

Figure 10: Dropping of Categorical Variables

Figure 11 below shows a printout to confirm no Categorical value was retained in the new data frame.

```
new_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 217149 entries, 0 to 217148
Data columns (total 23 columns):
 #   Column                          Non-Null Count   Dtype
---  ------                          --------------   -----
 0   POSTED_SPEED_LIMIT              217149 non-null  int64
 1   NUM_UNITS                       217149 non-null  int64
 2   INJURIES_TOTAL                  216607 non-null  float64
 3   INJURIES_NO_INDICATION          216607 non-null  float64
 4   CRASH_HOUR                      217149 non-null  int64
 5   CRASH_DAY_OF_WEEK               217149 non-null  int64
 6   CRASH_MONTH                     217149 non-null  int64
 7   LATITUDE                        215365 non-null  float64
 8   LONGITUDE                       215365 non-null  float64
 9   SEVERE_INJURY                   217149 non-null  int64
 10  POSTED_SPEED_LIMIT_CATEGORY     217149 non-null  category
 11  DAY_NIGHT_INDICATOR             217149 non-null  int64
 12  TRAFFICWAY_TYPE_NUMERIC         217149 non-null  int64
 13  ALIGNMENT_NUMERIC               217149 non-null  int64
 14  ROADWAY_SURFACE_NUMERIC         217149 non-null  int64
 15  ROAD_DEFECT_NUMERIC             217149 non-null  int64
 16  CRASH_TYPE_NUMERIC              217149 non-null  int64
 17  TRAFFIC_CONTROL_DEVICE_NUMERIC  217149 non-null  int64
 18  DEVICE_CONDITION_NUMERIC        217149 non-null  int64
 19  WEATHER_CONDITION_NUMERIC       217149 non-null  int64
 20  FIRST_CRASH_TYPE_NUMERIC        217149 non-null  int64
 21  DAMAGE_NUMERIC                  217149 non-null  int64
 22  PRIM_CONTRIBUTORY_CAUSE_NUMERIC 217149 non-null  int64
dtypes: category(1), float64(4), int64(18)
memory usage: 36.7 MB
```

Figure 11: Data frame info printout

Check for Null values in the data set was then carried out as shown in Figure 12 below.

Identified Null values were removed as shown in Figure 13.

```
#checking missing values in our new dataframe
new_df.isnull().sum().sort_values(ascending=False)
```

Figure 12: missing values check

```
columns_to_dropna = ['LATITUDE', 'LONGITUDE', 'INJURIES_TOTAL', 'INJURIES_NO_INDICATION']

new_df.dropna(subset=columns_to_dropna, inplace=True)
```

Figure 13: Deleting Null Values

3.4 Modeling and Evaluation

For this study, three models were built; Random Forest, Support Vector Machine, and Logistic Regression.

3.4.1 Experiment 1

A) Random Forest

The clean df dataset was used to define features and target variables, with columns assigned to predict crash severity. The dataset was split into training and testing sets, with 85% for training and 15% for testing. A Random Forest Classifier model was instantiated, with 100 decision trees and 42 random states for reproducibility. The model was trained using the training data.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix

# Define features (X) and target variable (y) for classification
features_cls = cleaned_df[['POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE_NUMERIC', 'DEVICE_CONDITION_NUMERIC',
                           'WEATHER_CONDITION_NUMERIC', 'DAY_NIGHT_INDICATOR', 'TRAFFICWAY_TYPE_NUMERIC', 'ALIGNMENT_NUMERIC',
                           'ROADWAY_SURFACE_NUMERIC', 'ROAD_DEFECT_NUMERIC', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH',
                           'LATITUDE', 'LONGITUDE'
                          ]]

target_cls = cleaned_df['CRASH_TYPE_NUMERIC']

# Split the data into training and testing sets
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(features_cls, target_cls, test_size=0.15, random_state=42)

# Initialize and fit the Random Forest Classifier model
rf_classifier = RandomForestClassifier(n_estimators=100, random_state=42)
rf_classifier.fit(X_train_cls, y_train_cls)

# Make predictions on the test set
y_pred_cls = rf_classifier.predict(X_test_cls)

# Evaluate the model using multiple metrics
accuracy = accuracy_score(y_test_cls, y_pred_cls)
precision = precision_score(y_test_cls, y_pred_cls, average='weighted')
recall = recall_score(y_test_cls, y_pred_cls, average='weighted')
f1 = f1_score(y_test_cls, y_pred_cls, average='weighted')
classification_rep = classification_report(y_test_cls, y_pred_cls)
conf_matrix = confusion_matrix(y_test_cls, y_pred_cls)

# Additional metrics: True Positive Rate, False Positive Rate, True Negative Rate, False Negative Rate
tn, fp, fn, tp = conf_matrix.ravel()
true_positive_rate = tp / (tp + fn)
false_positive_rate = fp / (fp + tn)
true_negative_rate = tn / (tn + fp)
false_negative_rate = fn / (fn + tp)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print("Classification Report:")
print(classification_rep)
print("Confusion Matrix:")
print(conf_matrix)
print(f"True Positive Rate: {true_positive_rate}")
print(f"False Positive Rate: {false_positive_rate}")
print(f"True Negative Rate: {true_negative_rate}")
print(f"False Negative Rate: {false_negative_rate}")
```

Figure 14: Random Forest Code

B) Support Vector Machine

The data is classified using features and target variables, similar to Random Forest. The dataset is split into training and testing sets using an 85:15 ratio. A Support Vector Machine Classifier (SVC) model is initialized and trained on the training data, setting hyperparameter values and ensuring reproducibility. The kernel was set to linear, gamma to auto while C, which is the regularisation parameter, was set to 1 as shown in Figure 15 below.

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split

# Define features (X) and target variable (y) for classification
features_cls = cleaned_df[['POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE_NUMERIC', 'DEVICE_CONDITION_NUMERIC',
                           'WEATHER_CONDITION_NUMERIC', 'DAY_NIGHT_INDICATOR', 'TRAFFICWAY_TYPE_NUMERIC', 'ALIGNMENT_NUMERIC',
                           'ROADWAY_SURFACE_NUMERIC', 'ROAD_DEFECT_NUMERIC', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH',
                           'LATITUDE', 'LONGITUDE'
                          ]]

target_cls = cleaned_df['CRASH_TYPE_NUMERIC']

# Split the data into training and testing sets
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(features_cls, target_cls, test_size=0.15, random_state=42)

# Define the Support Vector Machine classifier model
svm_classifier = SVC(C=1, kernel='linear', gamma='auto', random_state=42)

# Fit the model with hyperparameter tuning on the training data
svm_classifier.fit(X_train_cls, y_train_cls)

# Make predictions on the test set
y_pred_svm = svm_classifier.predict(X_test_cls)

# Evaluate the model using multiple metrics
accuracy_svm = accuracy_score(y_test_cls, y_pred_svm)
precision_svm = precision_score(y_test_cls, y_pred_svm, average='weighted')
recall_svm = recall_score(y_test_cls, y_pred_svm, average='weighted')
f1_svm = f1_score(y_test_cls, y_pred_svm, average='weighted')
classification_rep_svm = classification_report(y_test_cls, y_pred_svm)
conf_matrix_svm = confusion_matrix(y_test_cls, y_pred_svm)

# Additional metrics: True Positive Rate, False Positive Rate, True Negative Rate, False Negative Rate
tn_svm, fp_svm, fn_svm, tp_svm = conf_matrix_svm.ravel()
true_positive_rate_svm = tp_svm / (tp_svm + fn_svm)
false_positive_rate_svm = fp_svm / (fp_svm + tn_svm)
true_negative_rate_svm = tn_svm / (tn_svm + fp_svm)
false_negative_rate_svm = fn_svm / (fn_svm + tp_svm)

print("Support Vector Machine Metrics:")
print(f"Accuracy: {accuracy_svm}")
print(f"Precision: {precision_svm}")
print(f"Recall: {recall_svm}")
print(f"F1 Score: {f1_svm}")
print("Classification Report:")
print(classification_rep_svm)
print("Confusion Matrix:")
print(conf_matrix_svm)
print(f"True Positive Rate: {true_positive_rate_svm}")
print(f"False Positive Rate: {false_positive_rate_svm}")
print(f"True Negative Rate: {true_negative_rate_svm}")
print(f"False Negative Rate: {false_negative_rate_svm}")
```

Figure 15: Support Vector Machine Code with linear kernel and auto gamma

C) Logistic Regression

The logistic regression model was initialized and trained using the same process flow for data preprocessing and splitting, with hyperparameter configurations such as C for weaker regularization, max iter for convergence, and solver for optimization. Figure 16 shows the code for Logistic regression.

No specific hyperparameter values were set during the instantiation of the logistic regression model using the code entry: logistic_regression = LogisticRegression(). Con-

sequently, all hyperparameters retained their default settings, allowing the model to utilize the predefined configurations provided by the default parameters.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
import pandas as pd


# Define features (X) and target variable (y) for classification
features_cls = cleaned_df[['POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE_NUMERIC', 'DEVICE_CONDITION_NUMERIC',
                           'WEATHER_CONDITION_NUMERIC', 'DAY_NIGHT_INDICATOR', 'TRAFFICWAY_TYPE_NUMERIC', 'ALIGNMENT_NUMERIC',
                           'ROADWAY_SURFACE_NUMERIC', 'ROAD_DEFECT_NUMERIC','CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH',
                           'LATITUDE', 'LONGITUDE'
                          ]]

target_cls = cleaned_df['CRASH_TYPE_NUMERIC']

# Split the data into training and testing sets
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(features_cls, target_cls, test_size=0.15, random_state=42)

# Initialize and fit the Logistic Regression model
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train_cls, y_train_cls)

# Make predictions on the test set
y_pred_logistic = logistic_regression.predict(X_test_cls)

# Evaluate the model using multiple metrics
accuracy_logistic = accuracy_score(y_test_cls, y_pred_logistic)
precision_logistic = precision_score(y_test_cls, y_pred_logistic)
recall_logistic = recall_score(y_test_cls, y_pred_logistic)
f1_logistic = f1_score(y_test_cls, y_pred_logistic)
classification_rep_logistic = classification_report(y_test_cls, y_pred_logistic)
conf_matrix_logistic = confusion_matrix(y_test_cls, y_pred_logistic)

# Print the evaluation metrics
print("Logistic Regression Metrics:")
print(f"Accuracy: {accuracy_logistic}")
print(f"Precision: {precision_logistic}")
print(f"Recall: {recall_logistic}")
print(f"F1 Score: {f1_logistic}")
print("Classification Report:")
print(classification_rep_logistic)
print("Confusion Matrix:")
print(conf_matrix_logistic)
```

Figure 16: Logistic Regression Code

D) Performance Comparison

The performance comparison of experiment 1 is captured in Table 1 below:

Algorithm	Accuracy	Precision	Recall	F-1 Score
Random Forest	0.7105	0.6809	0.7105	0.6741
Support Vector Machine	0.6955	0.4837	0.6955	0.5705
Logistic Regression	0.6944	0.6995	0.9826	0.8172

Table 1: Experiment 1 Performance Comparison

3.4.2 Experiment 2

The study used GridSearchCV, a Scikit-Learn function, to identify optimal hyperparameter values for machine learning models. The study compared and selected the best values, infused them into the existing model code, and rerun the modified codes, resulting in improved performance and enhanced metric scores.

The second experiment aimed to improve model accuracy, robustness, and generalizability by examining the influence of these techniques on machine learning model performance. The final outputs of each model were compared to identify the best-fitting

algorithm for crash severity prediction.

A) Random Forest Figure 17 shows a code snippet incorporating the best-fit hyperparameter values in Random Forest code.

```
# Initialize and fit the Random Forest Classifier model
rf_classifier = RandomForestClassifier(n_estimators=150, max_depth=20, min_samples_leaf=4, min_samples_split=10, random_state=42
rf_classifier.fit(X_train_cls, y_train_cls)
```

Figure 17: Random Forest Best hyperparameters

B) Support Vector Machine Figure 18 shows a code snippet incorporating the best-fit hyperparameter values in the Support Vector Machine code.

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix

# Define features (X) and target variable (y) for classification
features_cls = cleaned_df[['POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE_NUMERIC', 'DEVICE_CONDITION_NUMERIC',
                           'WEATHER_CONDITION_NUMERIC', 'DAY_NIGHT_INDICATOR', 'TRAFFICWAY_TYPE_NUMERIC', 'ALIGNMENT_NUMERIC',
                           'ROADWAY_SURFACE_NUMERIC', 'ROAD_DEFECT_NUMERIC', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH',
                           'LATITUDE', 'LONGITUDE'
                          ]]

target_cls = cleaned_df['CRASH_TYPE_NUMERIC']

# Split the data into training and testing sets
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(features_cls, target_cls, test_size=0.15, random_state=42)

# Define the Support Vector Machine classifier model
svm_classifier = SVC(C=1, kernel='rbf', gamma='scale', random_state=42)  # Use fixed best-fit hyperparameters
```

Figure 18: Support Vector Machine Best hyperparameters

C) Logistic Regression Figure 19 shows a code snippet incorporating the best-fit hyperparameter values in the Logistic Regression code.

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
import pandas as pd

# Define features (X) and target variable (y) for classification
features_cls = cleaned_df[['POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE_NUMERIC', 'DEVICE_CONDITION_NUMERIC',
                           'WEATHER_CONDITION_NUMERIC', 'DAY_NIGHT_INDICATOR', 'TRAFFICWAY_TYPE_NUMERIC', 'ALIGNMENT_NUMERIC',
                           'ROADWAY_SURFACE_NUMERIC', 'ROAD_DEFECT_NUMERIC', 'CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH',
                           'LATITUDE', 'LONGITUDE'
                          ]]

target_cls = cleaned_df['CRASH_TYPE_NUMERIC']

# Split the data into training and testing sets
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(features_cls, target_cls, test_size=0.15, random_state=42)

# Initialize logistic regression with best hyperparameters
logistic_regression = LogisticRegression(C=1, max_iter=100, solver='lbfgs')
```

Figure 19: Logistic Regression Best hyperparameters

The final models' performance output is summarized below.

D) Final Algorithms Performance Comparison

The performance comparison of Experiment 2 is captured in Table 2 below:

Algorithm	Accuracy	Precision	Recall	F-1 Score
Random Forest	0.7178	0.6921	0.7178	0.6707
Support Vector Machine	0.6965	0.6703	0.6965	0.5758
Logistic Regression	0.6952	0.6994	0.9850	0.8180

Table 2: Experiment 2 Performance Comparison

3.5 Identifying Influential Factors

The study also investigated factors determining crash severity in Chicago using a random forest classifier. The classifier learned from the training set and calculated feature importance. The graphs analyzed the importance of each feature in determining crash severity, providing insight into factors affecting crash severity. Figure 20 shows the code snippet and output of the influential factors in ascending order.

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier

# Define features (X) and target variable (y)
features = cleaned_df[['POSTED_SPEED_LIMIT', 'TRAFFIC_CONTROL_DEVICE_NUMERIC', 'DEVICE_CONDITION_NUMERIC',
                       'WEATHER_CONDITION_NUMERIC', 'DAY_NIGHT_INDICATOR', 'TRAFFICWAY_TYPE_NUMERIC', 'ALIGNMENT_NUMERIC',
                       'ROADWAY_SURFACE_NUMERIC', 'ROAD_DEFECT_NUMERIC','CRASH_HOUR', 'CRASH_DAY_OF_WEEK', 'CRASH_MONTH',
                       'LATITUDE', 'LONGITUDE'
                      ]]
target = cleaned_df['CRASH_TYPE_NUMERIC']

# Initialize Random Forest classifier
rf = RandomForestClassifier(random_state=42)

# Fit the model
rf.fit(features, target)

# Get feature importances
feature_importances = rf.feature_importances_
feature_names = features.columns.tolist()

# Create a DataFrame of feature importances
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': feature_importances})

# Sort the features by importance in descending order
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Display feature importance
print(feature_importance_df)
```

```
                         Feature  Importance
12                      LATITUDE    0.226445
13                     LONGITUDE    0.223628
9                      CRASH_HOUR    0.135863
11                    CRASH_MONTH    0.106825
10             CRASH_DAY_OF_WEEK    0.077597
5          TRAFFICWAY_TYPE_NUMERIC    0.059934
0               POSTED_SPEED_LIMIT    0.036979
7          ROADWAY_SURFACE_NUMERIC    0.027995
1    TRAFFIC_CONTROL_DEVICE_NUMERIC    0.024069
3         WEATHER_CONDITION_NUMERIC    0.021033
2         DEVICE_CONDITION_NUMERIC    0.019948
8              ROAD_DEFECT_NUMERIC    0.019178
4             DAY_NIGHT_INDICATOR    0.012185
6               ALIGNMENT_NUMERIC    0.008322
```

Figure 20: Influential Factors

3.6 Sub-Factors Identification

The study focused on the primary contributory cause and crash type attributes in a dataset, using feature encoding and the One-Hot Encoding technique. The dataset was divided into training and testing sets, with 80% used for training the Random Forest Classifier. The classifier was trained to identify influential sub-factors impacting severe crashes, and

feature importance was extracted to identify the top 10 influential sub-factors. This information was visually represented in a horizontal bar plot using Matplotlib. Figure 21 shows the code snippet and output of the 10 most influential sub-factors in ascending order.

```
# Extract feature importance
feature_importance = pd.Series(random_forest.feature_importances_, index=encoded_df.columns)

# Sort feature importance in descending order
sorted_feature_importance = feature_importance.sort_values(ascending=False)

# Display top influential causes
print("Top Influential Causes for Severe Crashes:")
print(sorted_feature_importance.head(10))  # Display top 10 causes
```

```
Top Influential Causes for Severe Crashes:
PRIM_CONTRIBUTORY_CAUSE_DISREGARDING TRAFFIC SIGNALS                          0.188797
PRIM_CONTRIBUTORY_CAUSE_IMPROPER BACKING                                      0.110629
PRIM_CONTRIBUTORY_CAUSE_FAILING TO YIELD RIGHT-OF-WAY                         0.096491
PRIM_CONTRIBUTORY_CAUSE_UNABLE TO DETERMINE                                   0.065536
PRIM_CONTRIBUTORY_CAUSE_UNDER THE INFLUENCE OF ALCOHOL/DRUGS (USE WHEN ARREST IS EFFECTED)    0.064333
PRIM_CONTRIBUTORY_CAUSE_FAILING TO REDUCE SPEED TO AVOID CRASH                0.062934
PRIM_CONTRIBUTORY_CAUSE_FOLLOWING TOO CLOSELY                                 0.062273
PRIM_CONTRIBUTORY_CAUSE_PHYSICAL CONDITION OF DRIVER                          0.055161
PRIM_CONTRIBUTORY_CAUSE_IMPROPER OVERTAKING/PASSING                           0.054547
PRIM_CONTRIBUTORY_CAUSE_EQUIPMENT - VEHICLE CONDITION                         0.039366
dtype: float64
```

Figure 21: Influential Sub-Factors

3.7 Visualization of Influential Factors

Figure 22 shows the visualization output of the influential factors in ascending order.

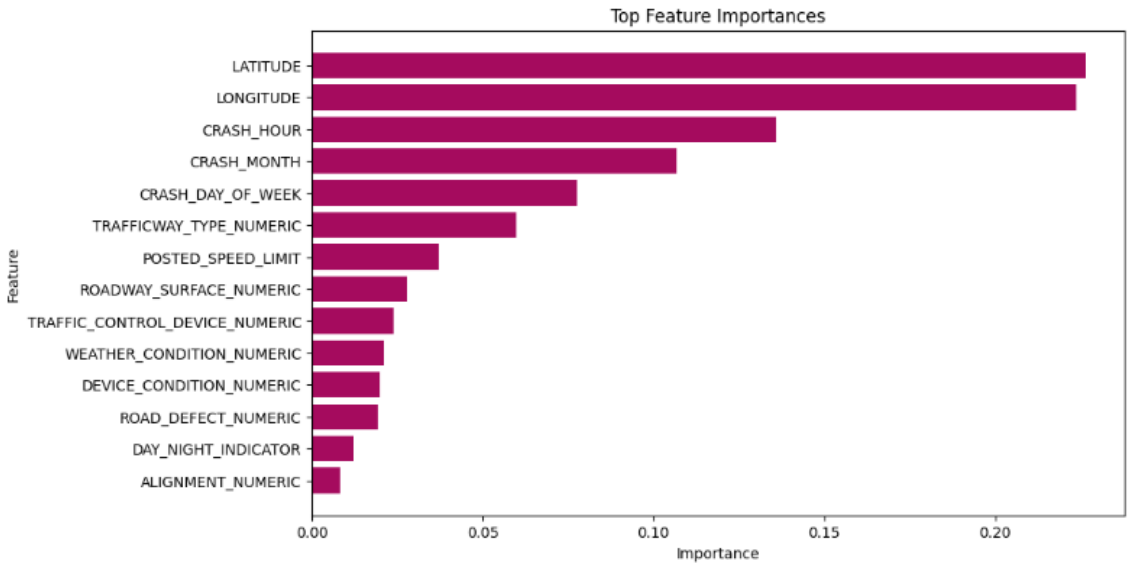

Figure 22: Visualization of Influential Factors