Configuration Manual

MSc Research Project
Programme Name

Saikrishnan Murali
Student ID: 20217200

School of Computing
National College of Ireland

Supervisor: Prashanth Nayak

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Saikrishnan Murali
Student ID:	20217200
Programme:	Programme Name
Year:	2022
Module:	MSc Research Project
Supervisor:	Prashanth Nayak
Submission Due Date:	15/08/2022
Project Title:	Configuration Manual
Word Count:	844
Page Count:	14

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:	
Date:	18th September 2022

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

Saikrishnan Murali
20217200

1 Introduction

This configuration manual is used to describe the requirements for the research project on **Navigation System to Avoid Accident Prone Zones using Machine Learning Techniques**. The research project, which comprises of three machine learning models and a navigation simulation using Open Route Service (ORS), will be discussed at each level so that the results may be replicated with exact results.

2 System Specification

The following is a list of the overall hardware and software requirements for replicating the project:

SYSTEM REQUIREMENT	SPECIFICATION
Processor	Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz 2.21 GHz
RAM	8.00 GB
Storage	512GB SSD
Operative System	Windows 11
Graphics hardware	NVIDIA GeForce GTX 1650 (4.0 GB)
System Type	64-bit operating system, x64-based processor
IDE	Jupter Notebook (Anaconda)
Open Source API Platform	OpenRouteService (ORS)
Web Browser	Google Chrome (Version 104.0.5112.81)

Table 1: System Specification and Requirement

The Hardware requirements mentioned above are more than sufficient for running the program. The Jupyter Notebook is an Integrated Development Environment (IDE) which is used as an end to end development of the project which is installed from the Anaconda Navigator. The OpenRouteService is a software application or a website developed using Open Street Map data that is used to provide route assistance via its API service. Section 3 explains the installation and setup process for these two.

3 Installations and Setup

This section will cover both the installation of Jupyter notebooks and the setting up of the OpenRouteService API.

3.1 Installation of Jupyter Notbook

To access the Jupyter Notebook, the Anaconda must be installed. Based on the type of OS (Windows) the anaconda needs to be installed. Once it is done the Anaconda Navigator window is opened to install and launch the Jupyter Notebook. Figures 1 and 2 explain how the Jupyter needs to be installed.

Figure 1: Installing Anaconda

The website for installation is in the footnote[1]

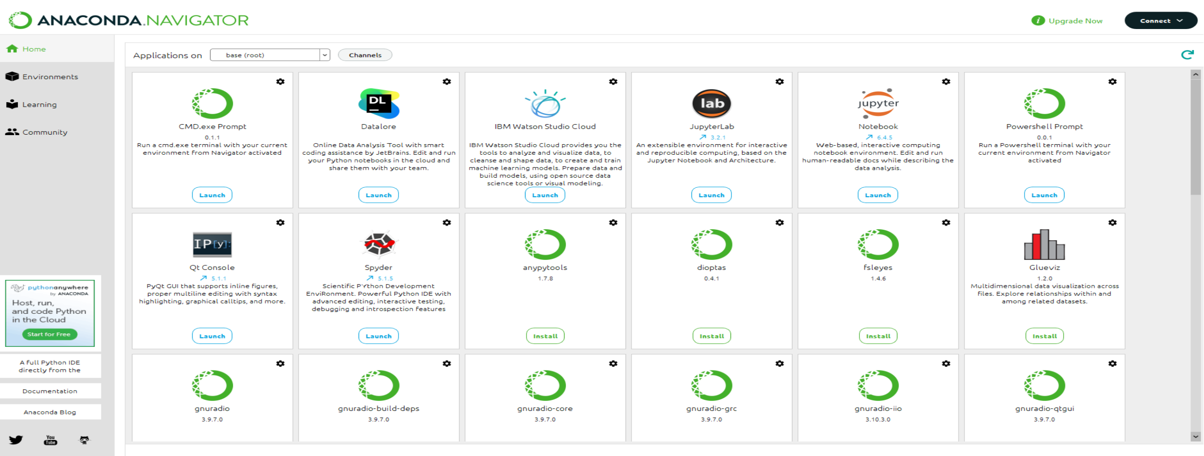

Figure 2: : Installing and Launching Jupyter

[1]https://www.anaconda.com/products/distribution

3.2 Setting up OpenRouteService API

The OpenRouteService (ORS) API must be configured to access its navigation and direction functions.

1. Sign up with ORS website using gmail account or Github Account (Figure 3)
2. Enter username and other creditial details and under Sector select as 'others' to avail the standard API service or the sector.(Figure 4 and 5).
3. If the sector is selected as 'other' select the standard option and give a the token name. (Figure 6)
4. Figure 7 shows the token name and the key details which will be used to access the API service.

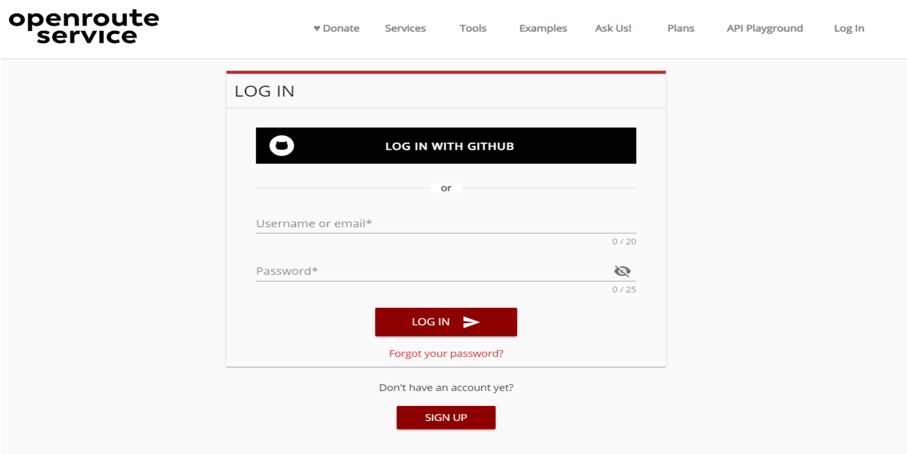

Figure 3: Sign Up with OpenRouteService (ORS)

The website for installation is in the footnote[2]

Figure 4: Selecting Sector details

[2]https://openrouteservice.org/dev/#/login

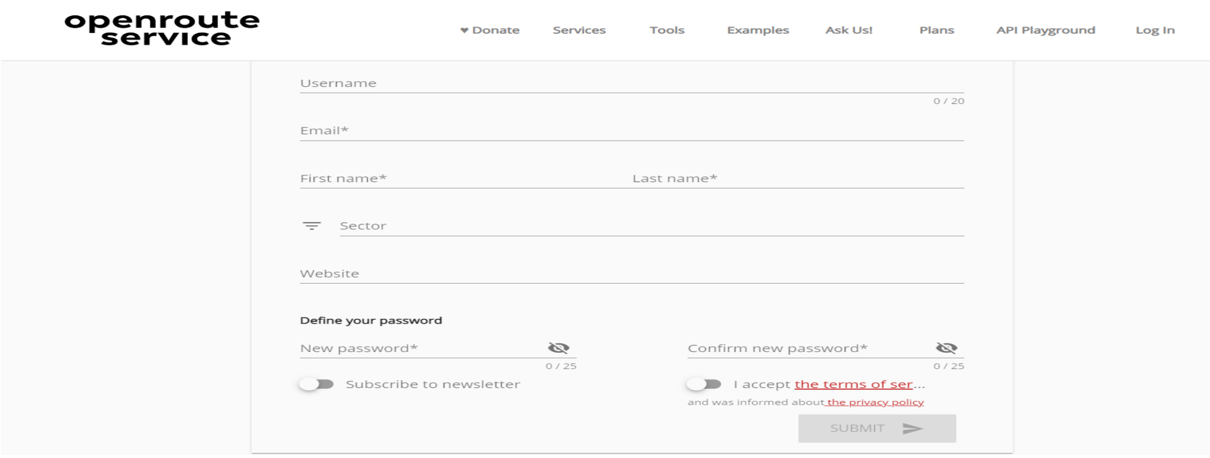

Figure 5: Entering the required details for API access

Figure 6: Requesting Token for API Key

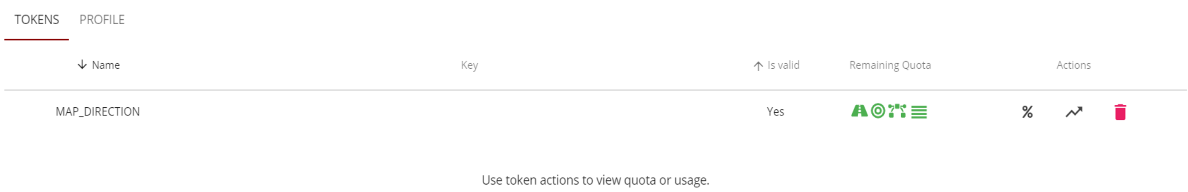

Figure 7: Using the API-KEY for Navigation Purpose

4 Data Source

Datasets related to road traffic accidents in Chicago will be used to develop a navigation system that will utilize Machine Learning techniques to avoid accident prone locations. These three datasets contain information about environment,vehicles and people involved in each accident that occur daily and they can be be acquired from the open Chicago repository[3].

5 Environment Setup and Package Installation

A new python file is opened using the Jupyter notebook which is launched from the Anaconda navigation window and opens in the web browser (Google Chrome) (Figure 8).The Python 3 option in the upper right corner of the Jupyter webpage is selected to run using the Python programming language (Figure 9).

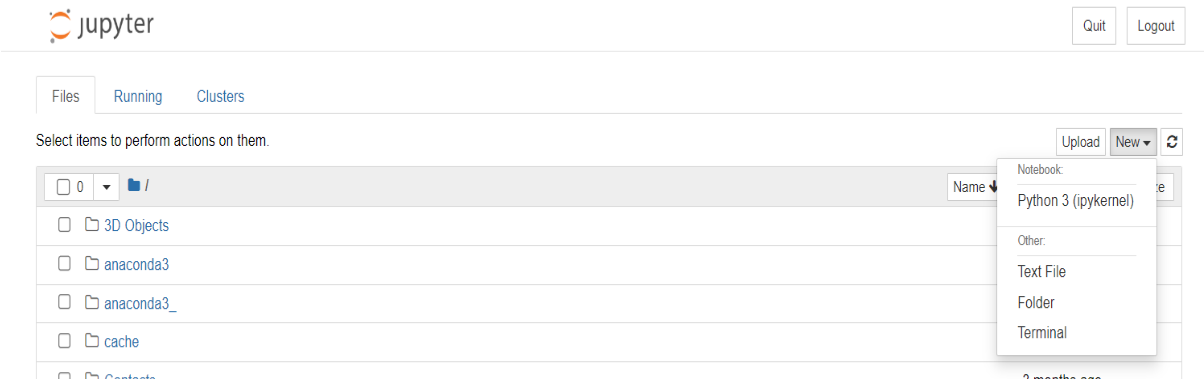

Figure 8: Open the Python Jupyter Notebook

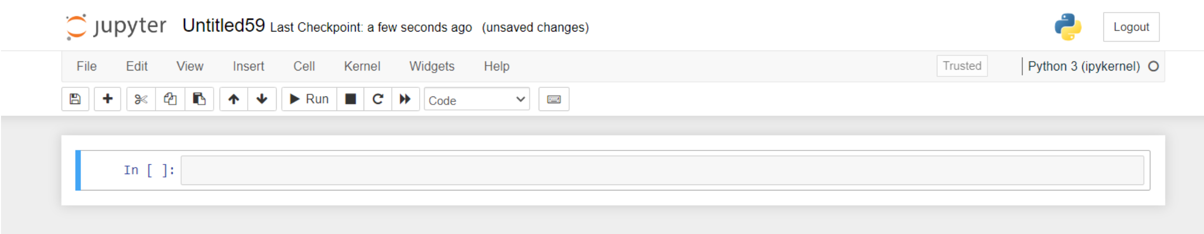

Figure 9: Python Execution Page

[3]https://data.cityofchicago.org/

```python
# Import Packages and Libraries required
import pandas as pd
import numpy as np
import tensorflow as tf
import folium
from folium import plugins
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import copy
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report,plot_confusion_matrix
from sklearn import metrics
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree
from sklearn.metrics import plot_roc_curve
from sklearn.metrics import roc_auc_score,roc_curve
from sklearn.model_selection import GridSearchCV
import shap
import statsmodels.api as sm
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
import math
from sklearn.metrics import mean_absolute_error
from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs
from itertools import cycle
import matplotlib.cm as cm
from geopy.geocoders import Nominatim
from geopy.distance import distance as gd
import pyproj
import requests
from openrouteservice import client
from shapely import geometry
from shapely.geometry import Point, LineString, Polygon, MultiPolygon
```

Figure 10: Libraries and Packages used in the project

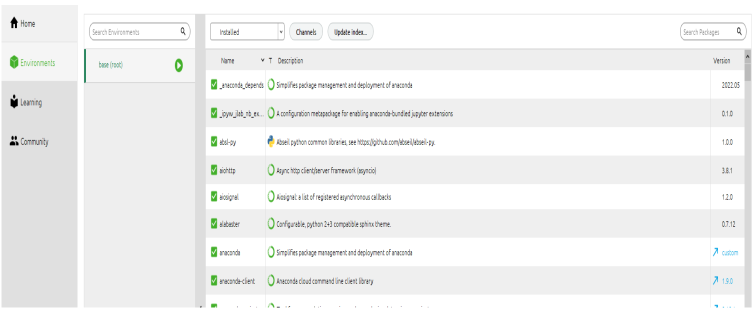

Figure 11: Installing packages using Anaconda Navigator

The Jupyter notebook comes with several packages installed by default, however in order to execute machine learning operations, additional packages and libraries must be loaded before utilizing their corresponding functions. The libraries and packages that needs to be installed are the ones in figure 10.The packages can be installed with the Anaconda Navigator by navigating to the environment tab on the left and searching for the required packages.

6 Model Implementaiotn

6.1 Importing and Pre-Processing the Data

The datasets specified in section 4 are imported into the jupyter notebook and will be combined using common columns. Because data for the year 2021 is required, it is filtered once the value of the appropriate column is converted to to datetime datatype.(Figure 12). The columns with more than 70% null values are removed and Performing pre-processing by Removing and renaming the columns and their values(Figure 13)

```
# Importing the data
crash_data = pd.read_csv('C:/Users/mural/Desktop/Traffic dataset/Updated ones/Traffic_Crashes_-_Crashes.csv',low_memory=False)
vehicle_data = pd.read_csv('C:/Users/mural/Desktop/Traffic dataset/Updated ones/Traffic_Crashes_-_Vehicles.csv',low_memory=False
people_data = pd.read_csv('C:/Users/mural/Desktop/Traffic dataset/Updated ones/Traffic_Crashes_-_People.csv',low_memory=False)
```

```
# Merging the common columns
info1 = np.intersect1d(people_data.columns,(np.intersect1d(crash_data.columns, vehicle_data.columns)))
info2 = np.intersect1d(crash_data.columns, vehicle_data.columns)
info3 = np.intersect1d(people_data.columns, vehicle_data.columns)
print(info1,info2,info3)
```

```
['CRASH_DATE' 'CRASH_RECORD_ID' 'RD_NO'] ['CRASH_DATE' 'CRASH_RECORD_ID' 'RD_NO'] ['CRASH_DATE' 'CRASH_RECORD_ID' 'RD_NO' 'VEHI
CLE_ID']
```

```
# Converting the CRASH_DATE column to datetime data type

crash_data['CRASH_DATE'] = pd.to_datetime(crash_data['CRASH_DATE'])
vehicle_data['CRASH_DATE'] = pd.to_datetime(vehicle_data['CRASH_DATE'])
people_data['CRASH_DATE'] = pd.to_datetime(people_data['CRASH_DATE'])
```

```
# The 2021 data is only filtered
Final_crash_data = crash_data[(crash_data['CRASH_DATE'] >= '2021-01-01') & (crash_data['CRASH_DATE'] <= '2021-12-31')]
Final_vehicle_data = vehicle_data[(vehicle_data['CRASH_DATE'] >= '2021-01-01') & (vehicle_data['CRASH_DATE'] <= '2021-12-31')]
Final_people_data = people_data[(people_data['CRASH_DATE'] >= '2021-01-01') & (people_data['CRASH_DATE'] <= '2021-12-31')]
```

```
# The values are merged using the common columns
merged = pd.merge(left=Final_vehicle_data, right = Final_crash_data,
                  left_on=['CRASH_RECORD_ID','RD_NO','CRASH_DATE'], right_on=["CRASH_RECORD_ID",'RD_NO','CRASH_DATE'])
merged_data = pd.merge(left=Final_people_data, right=merged,left_on = ['VEHICLE_ID','CRASH_RECORD_ID','RD_NO','CRASH_DATE'],
                       right_on=['VEHICLE_ID','CRASH_RECORD_ID','RD_NO','CRASH_DATE'])
```

Figure 12: Importing the Datasets and Merging

```
nulls = merged_data.isna().sum()
null_percent = nulls[nulls>0] / len(merged_data)
null_percent.to_frame('% Null').style.background_gradient(cmap='Blues')
```

AREA_05_I	0.863031
AREA_06_I	0.839248
AREA_07_I	0.862170
AREA_08_I	0.917549
AREA_09_I	0.908593
AREA_10_I	0.858805
AREA_11_I	0.702543
AREA_12_I	0.689673
AREA_99_I	0.878220
FIRST_CONTACT_POINT	0.036326
CMV_ID	0.989940
USDOT_NO	0.994749
CCMC_NO	0.998983

```
# extracting columns with missing values greater than 70%
Index_label = null_percent[null_percent>.70].index.tolist()
```

```
# Creating another variable which drops the columns in Index_label
merged_data_1 = merged_data.drop(columns = Index_label)
```

Figure 13: Data Cleaning

Feature Engineering and Changing columns Names

```
# Feature Engineering the column season using the month column
month_bins = [0,4,7,10,13]
label=('Winter','Spring','Summer','Fall')
month_binned = pd.cut(severity_accident['CRASH_MONTH'], month_bins, labels= label)
month_binned= month_binned.cat.as_unordered()
severity_accident['SEASON'] = month_binned
```

```
# Removing Further Columns by after feature engineering

severity_accident['STREET_NO'] = (severity_accident['STREET_NO']).astype(str)
severity_accident['STREET_DIRECTION'] = (severity_accident['STREET_DIRECTION']).astype(str)
severity_accident['STREET_NAME'] = (severity_accident['STREET_NAME']).astype(str)
```

```
# Removing after columns after creating address
severity_accident['ADDRESS'] = severity_accident[['STREET_NO', 'STREET_DIRECTION', 'STREET_NAME']].agg(' '.join, axis=1)
severity_accident.drop(['STREET_NO','STREET_DIRECTION','STREET_NAME'],axis=1, inplace = True)
```

Figure 14: Feature Engineering and grouping columns

Feature Engineering is done by adding new columns based on old column values.(Figure 14)

Converting variables into categorical as they need to be converted into Dummies

```
severity_accident_clean[['TRAFFIC_CONTROL_DEVICE','DEVICE_CONDITION', 'WEATHER_CONDITION',
         'LIGHTING_CONDITION', 'FIRST_CRASH_TYPE' , 'TRAFFICWAY_TYPE',
         'ROADWAY_SURFACE_COND', 'ALIGNMENT','SEASON',
         'ROAD_DEFECT', 'DAMAGE', 'PRIM_CONTRIBUTORY_CAUSE','UNIT_TYPE','VEHICLE_TYPE','SEX',
            'SAFETY_EQUIPMENT','AIRBAG_DEPLOYED','DRIVER_VISION','POSTED_SPEED_RANGE']] = severity_accident_clean[['TRAFFIC_CONT
         'LIGHTING_CONDITION', 'FIRST_CRASH_TYPE' , 'TRAFFICWAY_TYPE',
         'ROADWAY_SURFACE_COND', 'ALIGNMENT','SEASON',
         'ROAD_DEFECT', 'DAMAGE', 'PRIM_CONTRIBUTORY_CAUSE','UNIT_TYPE','VEHICLE_TYPE','SEX',
            'SAFETY_EQUIPMENT','AIRBAG_DEPLOYED','DRIVER_VISION','POSTED_SPEED_RANGE']].astype('category')
```

```
#creating the dummies
dummies_V2=['TRAFFIC_CONTROL_DEVICE','DEVICE_CONDITION', 'WEATHER_CONDITION',
         'LIGHTING_CONDITION', 'FIRST_CRASH_TYPE', 'TRAFFICWAY_TYPE',
         'ROADWAY_SURFACE_COND', 'ALIGNMENT','SEASON','SEX',
         'ROAD_DEFECT', 'DAMAGE', 'PRIM_CONTRIBUTORY_CAUSE','UNIT_TYPE','VEHICLE_TYPE',
            'SAFETY_EQUIPMENT','AIRBAG_DEPLOYED','DRIVER_VISION','POSTED_SPEED_RANGE']
severity_accident_clean_dm =  pd.get_dummies(severity_accident_clean, columns = dummies_V2, drop_first = False )
```

```
# Removing Further columns after feature engineering or they are no longer needed for analysis
drop_list = ['POSTED_SPEED_LIMIT', 'RD_NO','CRASH_RECORD_ID','INJURIES_UNKNOWN',
             'MOST_SEVERE_INJURY']
severity_accident_clean_dm.drop(columns = drop_list, inplace = True)
```

Figure 15: One Hot Encoding

One Hot Encoding is performed after converting values into binary categorical variables.(Figure 15)

6.2 Data Mining

The data is split into training and testing and then the classification, regression and clustering algorithm is performed

Model Implementation for Severity

```
# Define Input and Target Variables
Target = severity_accident_clean_dm['SEVERE']
# Removing 'Severe' as it is the output and other variables as it contains non numerical values
Input = severity_accident_clean_dm.drop(columns=['SEVERE','CRASH_DATE','LOCATION','ADDRESS'], axis=1)
```

```
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 100)
```

```
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(Input, Target, random_state=10)
print('Training data')
print(y_train.value_counts())
print('\nTesting data')
print(y_test.value_counts())
X_train
```

Figure 16: Splitting Of Data

1. Performing Logistic Regression Model, Decision Tree Classifier Model and Random Forest Classifier Mode using the training and testing data (Figure 17,18,19)

Logistic Regression

```
# Creating the Logistic classifier, and fitting it on the training data to make predictions on the testing data

logreg = LogisticRegression(C=1e10)
logreg.fit(X_train, y_train)
```

```
# Predicting tesing data using logreg

logreg_prediction = logreg.predict(X_test)
```

```
# probabilit for each values

pred_prob_log = logreg.predict_proba(X_test)[:,1]
pred_prob_log
```

```
#printing classifiction report of logistic regression

print(classification_report(y_test, logreg_prediction))
```

```
#printing the confusion matrix of Logistic regression
cm_log = confusion_matrix(y_test,logreg_prediction)
cm_log_norm = confusion_matrix(y_test,logreg_prediction, normalize='all')
print('Predicted Results \n',cm_log,'\n\n Normalized Predicted Results \n',cm_log_norm)
```

Figure 17: Logistic Regression

Decision Tree

```python
# Creating the Decision Tree classifier, and fitting it on the training data to make predictions on the testing data

dtree_class = DecisionTreeClassifier()
dtree_class.fit(X_train, y_train)
dtree_prediction = dtree_class.predict(X_test)
```

```python
print(classification_report(y_test, dtree_prediction))
```

```python
#printing the confusion matrix of Decision Tree Classifier
cm_dt = confusion_matrix(y_test,dtree_prediction)
cm_dt_norm = confusion_matrix(y_test,dtree_prediction, normalize='all')
print('Predicted Results \n',cm_dt,'\n\n Normalized Predicted Results \n',cm_dt_norm)
```

```python
# probabilit for each values

pred_prob_dt = dtree_class.predict_proba(X_test)[:,1]
pred_prob_dt
```

Figure 18: Decision Tree Classifier

Random Forest Classifier

```python
# Creating the Random Forest classifier, and fitting it on the training data to make predictions on the testing data

rf_class = RandomForestClassifier()
rf_class.fit(X_train,y_train)
```

```python
# Predicting tesing data using Random Forest

rf_class_prediction = rf_class.predict(X_test)
```

```python
# probabilit for each values

pred_prob_rf = rf_class.predict_proba(X_test)[:,1]
pred_prob_rf
```

Figure 19: Random Forest Classifier

2. The Risk score is obtained using Multiple Linear Regression and the K-means Clustering is performed using that. (Figure 20 and 21)

```
Accident_Parameters[['Risk_Parameter_A', 'Risk_Parameter_B', 'Risk_Parameter_C']] =Accident_Parameters[['Risk_Parameter_A', 'Risk
Accident_Parameters['RISK_SCORE'] = 2*Accident_Parameters['Risk_Parameter_A'] + 1.5*Accident_Parameters['Risk_Parameter_B'] + 1.5
Accident_Parameters['RISK_SCORE'] = Accident_Parameters['RISK_SCORE'].astype(int)
```

```
reg_input = Accident_Parameters[['Risk_Parameter_A','Risk_Parameter_B', 'Risk_Parameter_C']]
reg_output = Accident_Parameters['RISK_SCORE']
```

Performing Regression to determine the Risk Score

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn import linear_model
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(reg_input, reg_output, random_state=10)
```

```
linear_model = linear_model.LinearRegression()
linear_model.fit(X_train_reg,y_train_reg)
predicted_output = linear_model.predict(X_test_reg)
predicted_output
```

```
array([4.12804077, 2.91775521, 2.91775521, ..., 2.91775521, 2.91775521,
       1.29192955])
```

Figure 20: Multiple Linear Regression

```
n_clusters = 3
kmeans = KMeans(n_clusters = 3, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
centers = kmeans.cluster_centers_
#Clusters in DataFrame
Accident_Prone['CLUSTERS'] = y_kmeans +1 # to step up to group 1 to 3
labels = {1: "Cluster 1", 2: "Cluster 2", 0: "Cluster 3"}
colors = cycle(cm.tab10.colors)
plt.figure()
for i in range(n_clusters):
    # plot one cluster for each iteration
    color = next(colors)
    # find indeces corresponding to cluser i
    idx = y_kmeans == i
    # plot cluster
    plt.scatter(X[idx, 0], X[idx, 1], color=color, s=50, label=labels[i], alpha=0.25)
    # plot center
    plt.scatter(centers[i, 0], centers[i, 1], edgecolors="k", linewidth=2, color=color, s=200, alpha=1)

#Arranging the Legend
handles, labels = plt.gca().get_legend_handles_labels()
order = [1,2,0]
plt.legend([handles[idx] for idx in order],[labels[idx] for idx in order])
plt.title('OVERALL ACCIDENT LOCATIONS IN CHICAGO')
plt.xlabel('LATITUDE')
plt.ylabel('LONGITUDE');
```

Figure 21: K-Means Clustering

6.3 Creating Navigation System

The Navigation Model is created Based on the Clustering output and the normal and alternate route is obtained using ORS (Figure 22)

```
def style_function(color):
    return lambda feature: dict(color=color,
                                weight=3,
                                opacity=0.5)


map_params.update({'location': ([41.881832, -87.623177]),
                   'zoom_start': 11})
Desired_Nagiation_System = folium.Map(**map_params)


# Using the Normal Route Map which was done before and comparing with the risk free route

def style_function_B(feature):
    return {'opacity': 5,
            'weight': 3,
            'color':'RED'}

folium.features.GeoJson(data=route_normal,
                        name='Normal Route',
                        style_function=style_function_B,
                        overlay=True).add_to(Desired_Nagiation_System)


# Eliminating Risky areas which are not in the immediate surrounding of the route of interest
# With Buffer value of 0.009 degrees covers area aroud


def style_function_D(feature):
    return {'opacity': 5,
            'weight': 3,
            'color':'GREEN'}

route_buffer = LineString(route_normal['features'][0]['geometry']['coordinates']).buffer(0.009)
folium.features.GeoJson(data=geometry.mapping(route_buffer),
                        name='Route Buffer',
                        style_function=style_function_D,
                        overlay=True).add_to(Desired_Nagiation_System)



# Plot which construction sites fall into the buffer Polygon
sites_buffer_poly = []
for site_poly in sites_poly:
    poly = Polygon(site_poly)
    if route_buffer.intersects(poly):
        folium.features.Marker(list(reversed(poly.centroid.coords[0]))).add_to(Desired_Nagiation_System)
        sites_buffer_poly.append(poly)



# Route which avoids accident prone areas

# Request normal route between appropriate locations without construction sites
request_params = {'coordinates': list(coordinates),
                  'format_out': 'geojson',
                  'profile': 'driving-car',
                  'preference': 'shortest',
                  'instructions': 'true',}

request_params['options'] = {'avoid_polygons': geometry.mapping(MultiPolygon(sites_buffer_poly))}
route_detour = ors_client.directions(**request_params)

def style_function_C(feature):
    return {'opacity': 5,
            'weight': 3,
            'color':'BLUE'}
route_detour = ors_client.directions(**request_params)
folium.features.GeoJson(data=route_detour,
                        name='Risk Free Route',
                        style_function=style_function_C,
                        overlay=True).add_to(Desired_Nagiation_System)


folium.Marker(list(start_latlng),popup=coordinates[0],icon=folium.Icon(color='orange', icon='home'),color="blue").add_to(Desired_
folium.Marker(list(end_latlng),popup=coordinates[1],icon=folium.Icon(color='black', icon='home'),color="red").add_to(Desired_Nagi



folium.TileLayer('openstreetmap').add_to(Desired_Nagiation_System)
folium.TileLayer('Stamen Terrain').add_to(Desired_Nagiation_System)
folium.TileLayer('Stamen Toner').add_to(Desired_Nagiation_System)
folium.TileLayer('Stamen Water Color').add_to(Desired_Nagiation_System)
folium.TileLayer('cartodbpositron').add_to(Desired_Nagiation_System)
folium.TileLayer('cartodbdark_matter').add_to(Desired_Nagiation_System)
folium.LayerControl().add_to(Desired_Nagiation_System)


Desired_Nagiation_System
```

Figure 22: Navigation Model using ORS

The directions for travelling the route is mentioned in figure 23

```
# Alternative Risk Free Route


# distance in Miles and duration in Hours

print("The total distance travelled in the detour route is",route_detour['features'][0]['properties']['segments'][0]['distance']
print("The total Time Taken to travel in the detour route is",route_detour['features'][0]['properties']['segments'][0]['duration'


# Distances are in meters
# Timings are in seconds
print('DIRECTIONS: \n')
for index, i in enumerate(route_detour['features'][0]['properties']['segments'][0]['steps']):
    print(index+1, i, '\n')
```

Figure 23: Risk Free Direction

7 Evaluating the Model

```
import statsmodels.api as sm

# Adding Constant for Adjusted R-square

constant = sm.add_constant(X_test_reg)
result = sm.OLS(y_test_reg, constant).fit()
print('R-squared values =',result.rsquared,'\nAdjusted R-Squared Value =', result.rsquared_adj)
```

```
R-squared values = 0.9743006427311183
Adjusted R-Squared Value = 0.9742993373337866
```

```
from sklearn.metrics import mean_squared_error
import math
print('The Mean Squared Error Value is = ',mean_squared_error(y_test_reg, predicted_output))
print('The Root Mean Squared Error Value is = ',math.sqrt(mean_squared_error(y_test_reg, predicted_output)))
```

```
The Mean Squared Error Value is =  0.03941243123091162
The Root Mean Squared Error Value is =  0.19852564376148393
```

```
from sklearn.metrics import mean_absolute_error
print('The Mean Absolute Error Value is = ',mean_absolute_error(y_test_reg, predicted_output))
```

```
The Mean Absolute Error Value is =  0.16398049583160038
```

```
sns.set_theme(color_codes=True)
sns.regplot(x=y_test_reg,y=predicted_output,ci=None,scatter_kws={"color": "blue"}, line_kws={"color": "red"});
plt.xlabel('Actual');
plt.ylabel('Predicted');
```

Figure 24: Multiple Linear Regression Evaluation

```
X = Accident_Prone.iloc[:, 0:2].values
# Using the elbow method to find the optimal number of clusters
wcss = []
for i in range(1, 10):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', random_state = 42)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 10), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
```

```
Text(0, 0.5, 'WCSS')
```

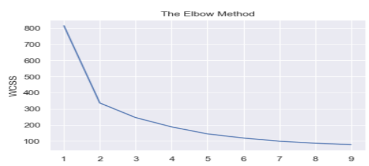

Figure 25: Evaluation of K-Means Clustering

The Multiple Linear Regression and Clustering models are evaluated as show in figure 24 and 25.The classifier models are all evaluated the same way and therefore Random Forest is evaluation is shown in figure 26

```
# Random Forest AUC and No Skill AUC

noskill = [0 for _ in range(len(y_test))]

Main_AUC_prob_rf = roc_auc_score(y_test, pred_prob_rf)
Noskill_AUC_rf = roc_auc_score(y_test, noskill)
print('Random Forest ROC AUC : %.3f' % Main_AUC_prob_rf)
print('No skill classifier ROC AUC: %.3f' % Noskill_AUC_rf)
print('Accuracy : ',metrics.accuracy_score(y_test, rf_class_prediction))
```

```
Random Forest ROC AUC : 0.980
No skill classifier ROC AUC: 0.500
Accuracy :  0.9347329213578262
```

```
fpr1, tpr1, thresholds1 = metrics.roc_curve(y_test_roc, pred_prob_log)
fpr2, tpr2, thresholds2 = metrics.roc_curve(y_test_roc, pred_prob_dt)
fpr3, tpr3, thresholds3 = metrics.roc_curve(y_test_roc, pred_prob_rf)

print("AUC score for Logistic Regression:", roc_auc_score(y_test, pred_prob_log,average='macro'))
print("AUC score for Decision Tree Classifier:", roc_auc_score(y_test, pred_prob_dt,average='macro'))
print("AUC score for Random Forest:", roc_auc_score(y_test, pred_prob_rf,average='macro'))

auc1 = metrics.roc_auc_score(y_test, pred_prob_log)
auc2 = metrics.roc_auc_score(y_test, pred_prob_dt)
auc3 = metrics.roc_auc_score(y_test, pred_prob_rf)

plt.plot(fpr1, tpr1,label="AUC="+str(auc1))
plt.plot(fpr2, tpr2,label="AUC="+str(auc2))
plt.plot(fpr3, tpr3,label="AUC="+str(auc3))


plt.plot([0,1], [0,1], ls=':')
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
```

```
#printing classification report of Random Forest
print('Classification Report for Random Forest Classifier \n')
print(classification_report(y_test, rf_class_prediction))
```

```
Classification Report for Random Forest Classifier

              precision    recall  f1-score   support

           0       0.92      0.99      0.95     38236
           1       0.97      0.84      0.90     20829

    accuracy                           0.93     59065
   macro avg       0.94      0.91      0.93     59065
weighted avg       0.94      0.93      0.93     59065
```

```
#printing the confusion matrix of Random Forest
cm_rf = confusion_matrix(y_test,rf_class_prediction)
print('Predicted Results \n',cm_rf)
mat_cm_rf = plot_confusion_matrix(rf_class,X_test,y_test);
mat_cm_rf;
```

Figure 26: Random Forest Evaluation