import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

import warnings

warnings.filterwarnings('ignore')

from plotly.offline import init\_notebook\_mode

init\_notebook\_mode(connected=True)

import os

for dirname, \_, filenames in os.walk('/kaggle/input'):

for filename in filenames:

print(os.path.join(dirname, filename))

data=pd.read\_csv("/Users/Nancy/Downloads/projectfolder/dataset.csv")

print(data.head())

print(data.info())

data.describe().T

numeric\_data = data.select\_dtypes(include=[np.number])

# Calculate skewness

skewness = numeric\_data.skew()

# Display skewness

print(skewness)

print(data.dtypes)

# Drop non-numeric columns or convert them if needed

numeric\_data = data.select\_dtypes(include=[np.number])

# Check for missing values

print(numeric\_data.isnull().sum())

# Calculate correlation

correlation = numeric\_data.corr()

# Plot correlation heatmap

sns.heatmap(correlation, annot=True, cmap='Blues')

plt.show()

fig=px.violin(data, data['Sex\_of\_driver'], data['Number\_of\_casualties'],template='ggplot2')

print(fig.show())

px.violin(data,data['Road\_surface\_type'], data['Number\_of\_casualties'],color='Accident\_severity',template='ggplot2')

df=data.copy(deep=True)

print(df.head())

print(df.isnull().sum())

print(df.shape)

print(df.size)

#REMOVING UNWANTED FEATURES, keeping required columns..

df.drop(['Time','Driving\_experience','Type\_of\_vehicle','Educational\_level'],axis=1,inplace=True)

df.drop(['Vehicle\_driver\_relation','Lanes\_or\_Medians','Owner\_of\_vehicle','Area\_accident\_occured','Road\_allignment',

'Types\_of\_Junction','Light\_conditions','Weather\_conditions','Vehicle\_movement','Fitness\_of\_casuality',

'Vehicle\_movement','Age\_band\_of\_driver','Sex\_of\_driver'],axis=1,inplace=True)

df.drop(['Pedestrian\_movement','Cause\_of\_accident','Work\_of\_casuality','Road\_surface\_conditions'],axis=1,inplace=True)

df.drop(['Service\_year\_of\_vehicle','Defect\_of\_vehicle'],axis=1,inplace=True)

df.dropna(inplace=True)

for i in df.columns:

if df[i].dtypes== object:

print(i)

print(df[i].unique())

print(df[i].nunique())

print()

print(df.head())

from sklearn.preprocessing import LabelEncoder

l = LabelEncoder()

for col in df.columns:

if df[col].dtype == object:

df[col] = l.fit\_transform(df[col])

print(df.head())

#MODEL BUILDING

x=df.drop('Accident\_severity',axis=1)

y=df['Accident\_severity']

from sklearn.model\_selection import train\_test\_split

xtrain, xtest, ytrain, ytest = train\_test\_split(x,y,test\_size=0.30)

from sklearn.preprocessing import MinMaxScaler

mms=MinMaxScaler(feature\_range=(0,1))

xtrain=mms.fit\_transform(xtrain)

xtest=mms.fit\_transform(xtest)

xtrain=pd.DataFrame(xtrain)

xtest=pd.DataFrame(xtest)

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import GridSearchCV

rf = RandomForestClassifier()

rf.fit(xtrain,ytrain)

ypred=rf.predict(xtest)

print('-----------------------------------------------------------------------------------------------------------------------')

from sklearn.metrics import confusion\_matrix, classification\_report, accuracy\_score, recall\_score, f1\_score, precision\_score

import pandas as pd

# Assuming ytest and ypred are your true labels and predictions

cm = confusion\_matrix(ytest, ypred)

cr = classification\_report(ytest, ypred, output\_dict=True) # Store as dictionary for DataFrame conversion

# Print confusion matrix with proper alignment

print("Confusion Matrix:")

cm\_df = pd.DataFrame(cm, index=["Actual 0", "Actual 1", "Actual 2"], columns=["Predicted 0", "Predicted 1", "Predicted 2"])

print(cm\_df)

print("\n")

# Convert classification report to DataFrame and round values to two decimals

cr\_df = pd.DataFrame(cr).transpose().round(2)

print("Classification Report:")

print(cr\_df)

print("\n")

# Calculate and print accuracy, precision, recall, and f1-score with two decimal places

accuracy = round(accuracy\_score(ytest, ypred), 2)

precision = round(precision\_score(ytest, ypred, average='weighted'), 2)

recall = round(recall\_score(ytest, ypred, average='weighted'), 2)

f1 = round(f1\_score(ytest, ypred, average='weighted'), 2)

# Print metrics with alignment

print(f"{'Metric':<15}{'Score':<10}")

print(f"{'-'\*25}")

print(f"{'Accuracy':<15}{accuracy:<10.2f}")

print(f"{'Precision':<15}{precision:<10.2f}")

print(f"{'Recall':<15}{recall:<10.2f}")

print(f"{'F1 Score':<15}{f1:<10.2f}")