Flood (2)

January 11, 2024

1 Appendix

1.0 Import Library

[1]:
```
# Data Manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from math import sqrt

import warnings
warnings.filterwarnings('ignore')
```

- NumPy (np): A Python package that provides functionality for doing numerical calculations.
- pandas (pd): A software library that enables users to manipulate and analyse data.
- Matplotlib.pyplot (plt): A Python package that enables the creation of static, animated, and interactive visualisations.
- sklearn.preprocessing.LabelEncoder: A tool that converts category information into numerical values.
- sklearn.preprocessing.StandardScaler: A tool used to standardise features by subtracting the mean and scaling to have a variance of one.
- sklearn.model_selection.train_test_split: Used to divide datasets into separate training and testing sets.
- imblearn.over_sampling.SMOTE: (Synthetic Minority Over-sampling Technique): A technique used to balance the class distribution in a dataset. It does this by producing synthetic samples specifically for the minority class.
- math.sqrt function: Used to compute the square root of a number.
- warnings module: To silence warning messages while the code is being executed.

2.0 Load the dataset

```
[2]: data = pd.read_csv('Rainfall_JPSTemerloh_2021_new.csv')
data1 = pd.read_csv('Rainfall_JPSTemerloh_2022_new.csv')
```

```
[3]: #make the 2021 data into one columns
newdata=data.melt('Day', value_name='Rainfall').drop('variable', axis=1)
print(newdata)
```

```
     Day Rainfall
0      1     26.0
1      2     36.0
2      3    125.0
3      4      3.5
4      5     36.0
..   …       …
367   27      0.0
368   28      0.0
369   29      0.0
370   30      2.0
371   31     29.0

[372 rows x 2 columns]
```

```
[4]: #make the 2022 data into one columns
newdata1=data1.melt('Day', value_name='Rainfall').drop('variable', axis=1)
print(newdata1)
```

```
     Day  Rainfall
0      1      44.5
1      2      25.0
2      3       0.5
3      4       8.0
4      5       0.5
..   …        …
367   27       0.0
368   28       0.0
369   29       0.0
370   30       0.0
371   31       0.0

[372 rows x 2 columns]
```

```
[5]: #Merge Rainfall Data
rainfalldata=pd.concat([newdata,newdata1], ignore_index=True)
print(rainfalldata)
```

```
     Day Rainfall
0      1     26.0
1      2     36.0
2      3    125.0
```

```
3       4      3.5
4       5     36.0
..    …       …
739    27      0.0
740    28      0.0
741    29      0.0
742    30      0.0
743    31      0.0

[744 rows x 2 columns]
```

[6]:

```python
data2 = pd.read_csv('WaterLevel_SgPahang_2021_new.csv')
data3 = pd.read_csv('WaterLevel_SgPahang_2022_new.csv')
```

[7]:

```python
#make the 2021 data into one columns
newdata2=data2.melt('Day', value_name='WaterLevel').drop('variable', axis=1)
print(newdata2)
```

```
     Day  WaterLevel
0      1       25.52
1      2       27.71
2      3       28.95
3      4       31.84
4      5       33.13
..   …          …
367   27        26.6
368   28       26.12
369   29       25.78
370   30       25.37
371   31       25.56

[372 rows x 2 columns]
```

[8]:

```python
#make the 2022 data into one columns
newdata3=data3.melt('Day', value_name='WaterLevel').drop('variable', axis=1)
print(newdata3)
```

```
     Day   WaterLevel
0      1        28.65
1      2        30.70
2      3        31.39
3      4        31.35
4      5        30.19
..   …            …
367   27        25.96
368   28        25.67
369   29        25.47
370   30        25.32
371   31        25.18
```

```
[372 rows x 2 columns]
```

[9]:

```
#Merge WaterLevel Data
waterleveldata=pd.concat([newdata2,newdata3], ignore_index=True)
print(waterleveldata)
```

```
     Day WaterLevel
0      1      25.52
1      2      27.71
2      3      28.95
3      4      31.84
4      5      33.13
..   …          …
739   27      25.96
740   28      25.67
741   29      25.47
742   30      25.32
743   31      25.18

[744 rows x 2 columns]
```

[10]:

```
data4 = pd.read_csv('Streamflow_SgPahang_2021_new.csv')
data5 = pd.read_csv('Streamflow_SgPahang_2022_new.csv')
```

[11]:

```
#make the 2021 data into one columns
newdata4=data4.melt('Day', value_name='StreamFlow').drop('variable', axis=1)
print(newdata4)
```

```
        Day  StreamFlow
0         1      719.41
1         2     2133.23
2         3     2999.71
3         4     4965.74
4         5     5845.11
..      …          …
391      29      849.70
392      30      644.58
393      31      745.48
394     NaN         NaN
395  Gap= 0         NaN

[396 rows x 2 columns]
```

[12]:

```
#make the 2022 data into one columns
newdata5=data5.melt('Day', value_name='StreamFlow').drop('variable', axis=1)
print(newdata5)
```

```
        Day  StreamFlow
```

```
0         1     2792.38
1         2     4204.10
2         3     4660.11
3         4     4634.53
4         5     3856.88
..       …         …
391      29      688.34
392      30      621.39
393      31      555.18
394     NaN         NaN
395  Gap= 0         NaN

[396 rows x 2 columns]
```

[13]:
```
#Merge StreamFlow Data
streamflowdata=pd.concat([newdata4,newdata5], ignore_index=True)
print(streamflowdata)
```

```
        Day  StreamFlow
0         1      719.41
1         2     2133.23
2         3     2999.71
3         4     4965.74
4         5     5845.11
..       …         …
787      29      688.34
788      30      621.39
789      31      555.18
790     NaN         NaN
791  Gap= 0         NaN

[792 rows x 2 columns]
```

[14]:
```
data6 = pd.read_csv('Weather_Temerloh_Celcius_2021.csv')
data7 = pd.read_csv('Weather_Temerloh_Celcius_2022.csv')
```

[15]:
```
#make the 2021 data into one columns
newdata6=data6.melt('Day', value_name='Weather').drop('variable', axis=1)
print(newdata6)
```

```
     Day    Weather
0      1  24.611111
1      2  24.333333
2      3  23.055556
3      4  23.388889
4      5  25.611111
..    …         …
367   27  27.055556
368   28  26.777778
```

```
369   29  26.722222
370   30  24.611111
371   31  25.666667

[372 rows x 2 columns]
```

[16]:
```
#make the 2022 data into one columns
newdata7=data7.melt('Day', value_name='Weather').drop('variable', axis=1)
print(newdata7)
```

```
     Day    Weather
0      1  25.444444
1      2  24.611111
2      3  25.055556
3      4  25.444444
4      5  27.000000
..   …          …
367   27  25.055556
368   28  25.722222
369   29  24.333333
370   30  26.444444
371   31  26.222222

[372 rows x 2 columns]
```

[17]:
```
#Merge Weather Data
weatherdata=pd.concat([newdata6,newdata7], ignore_index=True)
print(weatherdata)
```

```
     Day    Weather
0      1  24.611111
1      2  24.333333
2      3  23.055556
3      4  23.388889
4      5  25.611111
..   …          …
739   27  25.055556
740   28  25.722222
741   29  24.333333
742   30  26.444444
743   31  26.222222

[744 rows x 2 columns]
```

[18]:
```
#Merge All Data
waterleveldata1=waterleveldata.drop(columns=['Day'])
streamflowdata1=streamflowdata.drop(columns=['Day'])
weatherdata1=weatherdata.drop(columns=['Day'])
```

```
alldata=pd.concat([rainfalldata,waterleveldata1,streamflowdata1,weatherdata1],␣
↪axis=1)
print(alldata)
```

```
     Day Rainfall WaterLevel  StreamFlow    Weather
0    1.0     26.0      25.52      719.41  24.611111
1    2.0     36.0      27.71     2133.23  24.333333
2    3.0    125.0      28.95     2999.71  23.055556
3    4.0      3.5      31.84     4965.74  23.388889
4    5.0     36.0      33.13     5845.11  25.611111
..   …        …          …          …          …
787  NaN      NaN        NaN      688.34        NaN
788  NaN      NaN        NaN      621.39        NaN
789  NaN      NaN        NaN      555.18        NaN
790  NaN      NaN        NaN         NaN        NaN
791  NaN      NaN        NaN         NaN        NaN

[792 rows x 5 columns]
```

3.0 Data Preprocessing

[19]:
```
# Check for null data
alldata.isnull().sum()
```

[19]:
```
Day           48
Rainfall      62
WaterLevel    62
StreamFlow    62
Weather       62
dtype: int64
```

[20]:
```
alldata[pd.isnull(alldata).any(axis=1)]
```

[20]:
```
      Day Rainfall WaterLevel  StreamFlow    Weather
31    1.0     20.5      25.08         NaN  25.555556
32    2.0      0.0      25.14         NaN  26.055556
59   29.0      NaN        NaN      185.15        NaN
60   30.0      NaN        NaN      176.92        NaN
61   31.0      NaN        NaN         NaN        NaN
..    …        …          …          …          …
787   NaN      NaN        NaN      688.34        NaN
788   NaN      NaN        NaN      621.39        NaN
789   NaN      NaN        NaN      555.18        NaN
790   NaN      NaN        NaN         NaN        NaN
791   NaN      NaN        NaN         NaN        NaN

[118 rows x 5 columns]
```

Actually, there is no missing data, but since every month has a different number of days, it detects that as missing data.

[21]:
```
#Drop wrong day row missing data
alldata.dropna(subset = ['Weather'], inplace=True)
alldata[pd.isnull(alldata).any(axis=1)]
```

[21]:

	Day	Rainfall	WaterLevel	StreamFlow	Weather
31	1.0	20.5	25.08	NaN	25.555556
32	2.0	0.0	25.14	NaN	26.055556
62	1.0	0.0	?	NaN	27.444444
63	2.0	0.0	24.1	NaN	27.055556
64	3.0	0.0	24.15	NaN	27.222222
65	4.0	2.0	24.05	NaN	27.500000
97	5.0	2.0	24.6	NaN	26.555556
98	6.0	0.0	24.53	NaN	27.555556
129	6.0	0.5	25.13	NaN	26.833333
130	7.0	0.0	25.25	NaN	27.388889
131	8.0	0.0	25.3	NaN	28.166667
163	9.0	1.0	24.1	NaN	27.333333
164	10.0	1.5	24.04	NaN	26.777778
195	10.0	5.0	24.22	NaN	26.500000
196	11.0	4.5	24.35	NaN	25.500000
197	12.0	35.0	26.21	NaN	26.555556
229	13.0	0.0	23.84	NaN	28.222222
230	14.0	0.0	23.76	NaN	28.222222
262	15.0	1.5	24.2	NaN	26.444444
263	16.0	0.0	24.03	NaN	27.833333
294	16.0	0	23.37	NaN	29.388889
295	17.0	9.5	23.33	NaN	26.277778
296	18.0	0	23.34	NaN	27.555556
328	19.0	12.0	27.16	NaN	26.277778
329	20.0	37.5	26.56	NaN	27.722222
360	20.0	0.0	33.56	NaN	25.944444
361	21.0	1.5	34.34	NaN	26.888889
362	22.0	41.0	34.59	NaN	26.388889
394	23.0	0.0	24.53	NaN	26.777778
395	24.0	1.0	24.48	NaN	26.555556
427	25.0	0.5	24.24	NaN	23.833333
428	26.0	0.0	25.54	NaN	25.722222
457	24.0	0.0	24.78	NaN	27.666667
458	25.0	0.0	25.21	NaN	28.333333
459	26.0	1.0	25.34	NaN	27.944444
460	27.0	0.0	25.25	NaN	26.611111
461	28.0	0.0	25.48	NaN	27.500000
493	29.0	21.5	24.81	NaN	28.055556
494	30.0	0.0	24.81	NaN	27.666667
525	30.0	0.0	24.25	NaN	28.500000

```
526  31.0    1.0    24.13        NaN  28.333333
527   1.0    2.0    24.07        NaN  26.944444
559   2.0    0.0     24.3        NaN  29.111111
560   3.0    0.0    24.01        NaN  29.444444
591   3.0    2.0    23.93        NaN  26.333333
592   4.0    1.0     24.4        NaN  26.166667
593   5.0    0.0    24.43        NaN  27.055556
625   6.0    0.5    24.62        NaN  27.500000
626   7.0   20.5    24.33        NaN  26.500000
658   8.0    3.5    24.97        NaN  25.833333
659   9.0   32.0    26.34        NaN  26.666667
690   9.0    0.0    26.65        NaN  25.222222
691  10.0    4.0    26.39        NaN  25.777778
692  11.0    9.0    25.93        NaN  25.555556
724  12.0    1.0    28.71        NaN  25.277778
725  13.0   13.0    28.62        NaN  24.333333
```

[22]:

```
#WaterLevel Interpolation
#Merge Feb available data & March first 9 days data for interpolation
FebDay=alldata.iloc[31:40, 0]
MarDay=alldata.iloc[60:68,0]

daydata=pd.concat([FebDay,MarDay], ignore_index=True)
print(daydata)
```

```
0     1.0
1     2.0
2     3.0
3     4.0
4     5.0
5     6.0
6     7.0
7     8.0
8     9.0
9     2.0
10    3.0
11    4.0
12    5.0
13    6.0
14    7.0
15    8.0
16    9.0
Name: Day, dtype: float64
```

[23]:

```
#Merge Feb available data & March first 9 days data for interpolation
FebWaterLevel=alldata.iloc[31:40, 2]
MarWaterLevel=alldata.iloc[60:68,2]
#print(MarDay)
```

```
interpolatedata=pd.concat([FebWaterLevel,MarWaterLevel], ignore_index=True)
print(interpolatedata)
```

```
0     25.08
1     25.14
2     25.25
3     25.12
4     24.93
5     24.78
6     24.71
7     24.69
8     24.67
9      24.1
10    24.15
11    24.05
12    23.99
13    23.95
14       24
15    24.07
16    23.99
Name: WaterLevel, dtype: object
```

[24]:
```
#Linear Interpolation for missing value in Water Level
import scipy.interpolate

x=daydata
y=interpolatedata
y_interp = scipy.interpolate.interp1d(x, y)

#find y-value associated with x-value of 8.5 which is in between February and
↪March data
print(y_interp(8.5))
```

```
24.37
```

[25]:
```
#Replace missing data with Linear Interpolation value
alldata.iloc[40:60, 2] = y_interp(8.5)
print(alldata.iloc[40:60, 2])
```

```
40    24.37
41    24.37
42    24.37
43    24.37
44    24.37
45    24.37
46    24.37
47    24.37
48    24.37
49    24.37
```

```
50    24.37
51    24.37
52    24.37
53    24.37
54    24.37
55    24.37
56    24.37
57    24.37
58    24.37
62    24.37
Name: WaterLevel, dtype: object
```

[26]:
```
#Use this mask to filter and detect the question mark symbol (?)
def contains_question_mark(cell):
    return '?' in str(cell)

# Apply the function to the entire DataFrame
mask = alldata.applymap(contains_question_mark)
```

[27]:
```
rows_with_question_mark = alldata[mask.any(axis=1)]
rows_with_question_mark
```

[27]:

	Day	Rainfall	WaterLevel	StreamFlow	Weather
289	11.0	?	23.72	166.59	28.833333

[28]:
```
count_question_marks = mask.values.sum()
count_question_marks
```

[28]: 1

[29]:
```
alldata.replace('?', np.nan, inplace=True)
```

[30]:
```
#Replace missing rainfall data with 0
alldata.fillna(0, inplace = True)
```

[31]:
```
#Recheck for missing value
alldata.isnull().sum()
```

[31]:
```
Day           0
Rainfall      0
WaterLevel    0
StreamFlow    0
Weather       0
dtype: int64
```

[32]:
```
#export the complete dataset to csv
alldata.to_csv('AllData.csv', index=False)
```

[33]:
```
# Import the complete dataset
newdata = pd.read_csv('AllData.csv')
```

[34]:
```
newdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 730 entries, 0 to 729
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Day         730 non-null    float64
 1   Rainfall    730 non-null    float64
 2   WaterLevel  730 non-null    float64
 3   StreamFlow  730 non-null    float64
 4   Weather     730 non-null    float64
dtypes: float64(5)
memory usage: 28.6 KB
```

[35]:
```
# Define the threshold levels based on your domain knowledge
thresholds = [26.00, 29.00, 31.00]

# Function to classify water levels into thresholds
def classify_water_level(water_level):
    if water_level < thresholds[0]:
        return 'normal'
    elif thresholds[0] <= water_level < thresholds[1]:
        return 'alert'
    elif thresholds[1] <= water_level < thresholds[2]:
        return 'warning'
    else:
        return 'danger'

# Add a new column with the threshold labels
newdata['Threshold'] = newdata['WaterLevel'].apply(classify_water_level)
```

Since this research is about classification, so need to classify the water level by certain ranges to represent the normal, alert, warning and danger.

[36]:
```
newdata
```

[36]:

	Day	Rainfall	WaterLevel	StreamFlow	Weather	Threshold
0	1.0	26.0	25.52	719.41	24.611111	normal
1	2.0	36.0	27.71	2133.23	24.333333	alert
2	3.0	125.0	28.95	2999.71	23.055556	alert
3	4.0	3.5	31.84	4965.74	23.388889	danger
4	5.0	36.0	33.13	5845.11	25.611111	danger
..	…	…	…	…	…	…
725	27.0	0.0	25.96	755.32	25.055556	normal

```
726  28.0        0.0        25.67      712.40  25.722222    normal
727  29.0        0.0        25.47      699.56  24.333333    normal
728  30.0        0.0        25.32      789.36  26.444444    normal
729  31.0        0.0        25.18     1367.53  26.222222    normal

[730 rows x 6 columns]
```

[37]:
```python
#Describe the data
newdata.describe()
```

[37]:

	Day	Rainfall	WaterLevel	StreamFlow	Weather
count	730.000000	730.000000	730.000000	730.000000	730.000000
mean	15.720548	6.372603	25.188726	618.798493	27.051522
std	8.802278	15.945107	1.684204	977.532332	1.211477
min	1.000000	0.000000	23.330000	0.000000	23.055556
25%	8.000000	0.000000	24.192500	163.455000	26.277778
50%	16.000000	0.000000	24.720000	329.070000	27.166667
75%	23.000000	5.500000	25.677500	685.850000	27.944444
max	31.000000	165.000000	34.590000	6977.240000	29.888889

4.0 Data Observation & Visualization

[38]:
```python
# Label Encoding
lab = LabelEncoder()
newdata['Threshold'] = lab.fit_transform(newdata['Threshold'])
```

[39]:
```python
#Normalize value
from sklearn import preprocessing
d = preprocessing.normalize(newdata)
scaled_df = pd.DataFrame(d, columns=newdata.columns)
print(scaled_df)
```

```
          Day  Rainfall  WaterLevel  StreamFlow   Weather  Threshold
0    0.001387  0.036073    0.035407    0.998133  0.034146   0.002775
1    0.000937  0.016871    0.012986    0.999708  0.011403   0.000000
2    0.000999  0.041631    0.009642    0.999057  0.007679   0.000000
3    0.000805  0.000705    0.006412    0.999968  0.004710   0.000201
4    0.000855  0.006159    0.005668    0.999955  0.004381   0.000171
..        …         …          …          …         …          …
725  0.035683  0.000000    0.034308    0.998222  0.033113   0.002643
726  0.039222  0.000000    0.035959    0.997929  0.036032   0.002802
727  0.041367  0.000000    0.036331    0.997876  0.034710   0.002853
728  0.037937  0.000000    0.032019    0.998204  0.033441   0.002529
729  0.022655  0.000000    0.018402    0.999389  0.019163   0.001462

[730 rows x 6 columns]
```

[40]:
```python
newdata.corr()
```

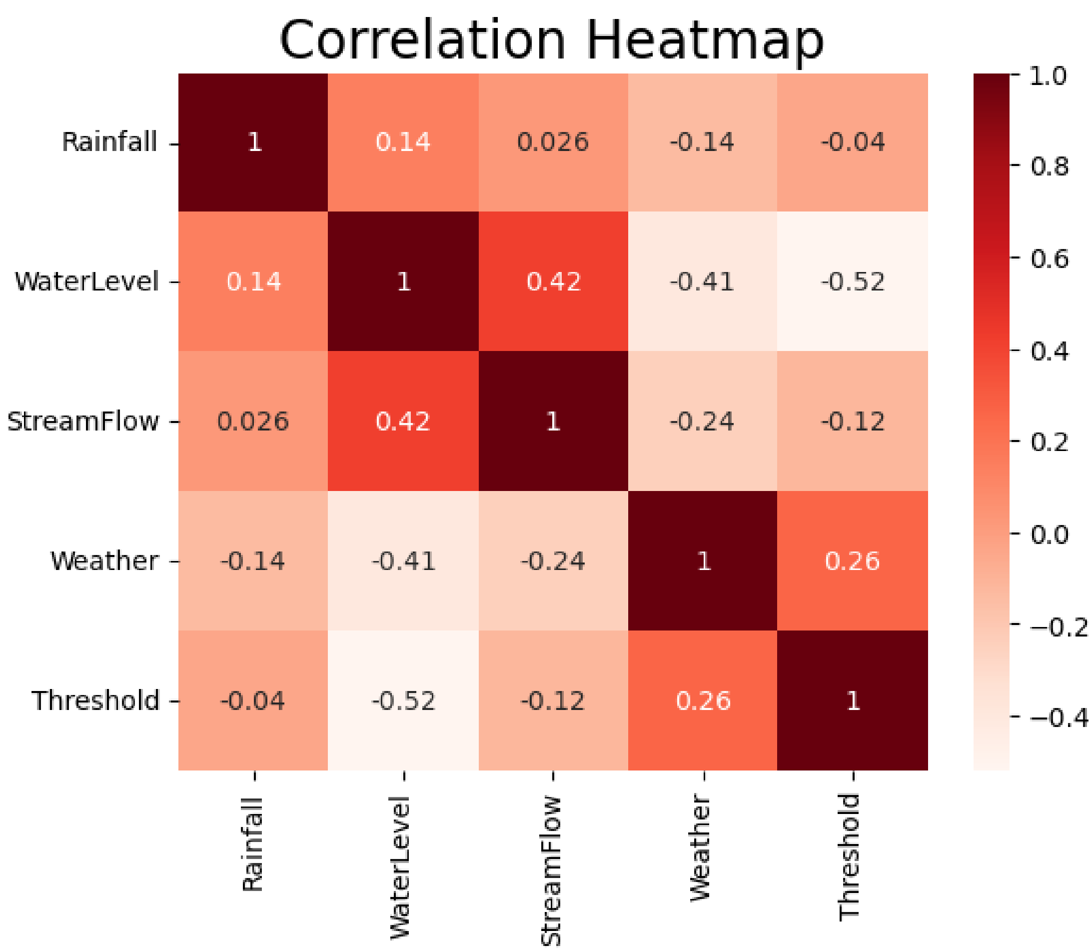
[40]:

	Day	Rainfall	WaterLevel	StreamFlow	Weather	Threshold
Day	1.000000	-0.072206	-0.040299	-0.020397	0.064462	0.019794
Rainfall	-0.072206	1.000000	0.139586	0.025883	-0.135205	-0.039510
WaterLevel	-0.040299	0.139586	1.000000	0.417328	-0.405941	-0.519297
StreamFlow	-0.020397	0.025883	0.417328	1.000000	-0.243720	-0.118819
Weather	0.064462	-0.135205	-0.405941	-0.243720	1.000000	0.260540
Threshold	0.019794	-0.039510	-0.519297	-0.118819	0.260540	1.000000

[41]:

```python
# see correlation between variables through a correlation heatmap
import seaborn as sns

#drop 'Day'
newdata1=newdata.drop(columns=['Day'],axis=1)

corr = newdata1.corr()
plt.figure()
sns.heatmap(corr, annot=True, cmap="Reds")
plt.title('Correlation Heatmap', fontsize=20)
```

[41]: Text(0.5, 1.0, 'Correlation Heatmap')



From the correlation heatmap, Streamflow has slight strong relation with Water Level (0.42).

While Threshold and Weather has slight strong negative relation with Water Level -0.52 and -0.41 respectively

5.0 Assign input and output variables

[42]:
```
# Assign feature and target
X = newdata[['Rainfall', 'StreamFlow', 'Weather']]
y_regression = newdata1['WaterLevel']
y_classification = newdata1['Threshold']
```

The feature and target variables are essential elements of the training data that the model utilises to acquire patterns and provide predictions.

The feature variables (X) are 'Rainfall', 'StreamFlow' and 'Weather', while the target variable for regression (y_regression) is 'WaterLevel'. Meanwhile, the target variable for classification (y_classification) represents the 'Threshold' labels, including 'normal', 'alert', 'warning' and 'danger' based on the water level.

6.0 Split data for train & test

[43]:
```
# Split the data into training and testing sets

X_train, X_test, y_reg_train, y_reg_test, y_class_train, y_class_test = 
 ↪train_test_split(
    X, y_regression, y_classification, test_size=0.2, random_state=42)
```

For data partitioning, the dataset is split into 20% for testing and the remaining 80% for training.

[44]:
```
# Feature Scaling
scaler = StandardScaler()
X_train_scale = scaler.fit_transform(X_train)
X_test_scale = scaler.fit_transform(X_test)
```

Feature scaling is significant to get better performance of machine learning models. Better machine learning may be distinguished from weaker machine learning via scaling. Standard scalar standardizes the features by eliminating the mean and scaling to unit variance. It also known as Z-score normalisation. It aims to convert the data into a standard normal distribution (with a mean of 0 and a standard deviation of 1).

[45]:
```
# Instantiate the label encoder
label_encoder = LabelEncoder()

# Fit and transform the target variable in the training set
y_class_train_encoded = label_encoder.fit_transform(y_class_train)

# Transform the target variable in the test set
```

```
y_class_test_encoded = label_encoder.transform(y_class_test)
```

Label encoder is used to convert the categorical labels such as 'normal', 'alert', 'warning', and 'danger' into numerical labels.

7.0 Model

1.1 Decision Tree (Classification)

[46]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Create a decision tree model for classification
dt_classifier = DecisionTreeClassifier(random_state=42)

# Fit the model on the training data
dt_classifier.fit(X_train, y_class_train)

# Predict on the test set
y_class_pred_dt = dt_classifier.predict(X_test)
```

1.2 Support Vector Machines (Classification)

[47]:
```
from sklearn.svm import SVC

# Create a support vector machine model for classification
svm_classifier = SVC(random_state=42)

# Fit the model on the training data
svm_classifier.fit(X_train, y_class_train)

# Predict on the test set
y_class_pred_svm = svm_classifier.predict(X_test)
```

1.3 Random Forest (Classification)

[48]:
```
from sklearn.ensemble import RandomForestClassifier

# Create a random forest model for classification
rf_classifier = RandomForestClassifier(random_state=42)

# Fit the model on the training data
rf_classifier.fit(X_train, y_class_train)

# Predict on the test set
y_class_pred_rf = rf_classifier.predict(X_test)
```

1.4 Gradient Boosting (Classification)

[49]:
```
from sklearn.ensemble import GradientBoostingClassifier

# Create a gradient boosting model for classification
gb_classifier = GradientBoostingClassifier(random_state=42)

# Fit the model on the training data
gb_classifier.fit(X_train, y_class_train)

# Predict on the test set
y_class_pred_gb = gb_classifier.predict(X_test)
```

1.5 XGBoost (Classification)

[50]:
```
import xgboost as xgb

# Define the XGBoost classifier
xgb_classifier = xgb.XGBClassifier(objective='multi:softmax',␣
 ↪num_class=len(set(y_classification)))

# Train the model on the training set
xgb_classifier.fit(X_train, y_class_train)

# Predict on the test set
y_class_pred_xgb = xgb_classifier.predict(X_test)
```

8.0 Model Evaluation

[51]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix ,mean_squared_error

# Evaluate Decision Tree model
accuracy_dt = accuracy_score(y_class_test, y_class_pred_dt)
classification_rep_dt = classification_report(y_class_test, y_class_pred_dt)
conf_matrix_dt = confusion_matrix(y_class_test, y_class_pred_dt)

print("Decision Tree Model Evaluation:")
print(f'Accuracy: {accuracy_dt:.4f}')
print('\nClassification Report:')
print(classification_rep_dt)
print('\nConfusion Matrix:')
print(conf_matrix_dt)

# Evaluate Random Forest model
accuracy_rf = accuracy_score(y_class_test, y_class_pred_rf)
classification_rep_rf = classification_report(y_class_test, y_class_pred_rf)
conf_matrix_rf = confusion_matrix(y_class_test, y_class_pred_rf)
```

```python
print("\nRandom Forest Model Evaluation:")
print(f'Accuracy: {accuracy_rf:.4f}')
print('\nClassification Report:')
print(classification_rep_rf)
print('\nConfusion Matrix:')
print(conf_matrix_rf)

# Evaluate SVM model
accuracy_svm = accuracy_score(y_class_test, y_class_pred_svm)
classification_rep_svm = classification_report(y_class_test, y_class_pred_svm)
conf_matrix_svm = confusion_matrix(y_class_test, y_class_pred_svm)

print("\nSupport Vector Machine Model Evaluation:")
print(f'Accuracy: {accuracy_svm:.4f}')
print('\nClassification Report:')
print(classification_rep_svm)
print('\nConfusion Matrix:')
print(conf_matrix_svm)

# Evaluate Gradient Boosting model
accuracy_gb = accuracy_score(y_class_test, y_class_pred_gb)
classification_rep_gb = classification_report(y_class_test, y_class_pred_gb)
conf_matrix_gb = confusion_matrix(y_class_test, y_class_pred_gb)

print("\nGradient Boosting Model Evaluation:")
print(f'Accuracy: {accuracy_gb:.4f}')
print('\nClassification Report:')
print(classification_rep_gb)
print('\nConfusion Matrix:')
print(conf_matrix_gb)

# Evaluate XGBoost model
accuracy_xgb = accuracy_score(y_class_test, y_class_pred_xgb)
classification_rep_xgb = classification_report(y_class_test, y_class_pred_xgb)
conf_matrix_xgb = confusion_matrix(y_class_test, y_class_pred_xgb)

print(f'XGBoost Classifier - Accuracy: {accuracy_xgb:.4f}')
print('\nClassification Report:')
print(classification_rep_xgb)
print('\nConfusion Matrix:')
print(conf_matrix_xgb)
```

```
Decision Tree Model Evaluation:
Accuracy: 0.7534

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

```
           0       0.32      0.35      0.33        23
           1       0.50      0.20      0.29         5
           2       0.86      0.86      0.86       117
           3       0.00      0.00      0.00         1

    accuracy                           0.75       146
   macro avg       0.42      0.35      0.37       146
weighted avg       0.76      0.75      0.75       146


Confusion Matrix:
[[  8   0  13   2]
 [  2   1   2   0]
 [ 15   1 101   0]
 [  0   0   1   0]]

Random Forest Model Evaluation:
Accuracy: 0.8288

Classification Report:
              precision    recall  f1-score   support

           0       0.60      0.26      0.36        23
           1       1.00      0.20      0.33         5
           2       0.84      0.97      0.90       117
           3       0.00      0.00      0.00         1

    accuracy                           0.83       146
   macro avg       0.61      0.36      0.40       146
weighted avg       0.81      0.83      0.79       146


Confusion Matrix:
[[  6   0  17   0]
 [  1   1   3   0]
 [  3   0 114   0]
 [  0   0   1   0]]

Support Vector Machine Model Evaluation:
Accuracy: 0.8014

Classification Report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00        23
           1       0.50      0.20      0.29         5
           2       0.81      0.99      0.89       117
```

```
           3       0.00      0.00      0.00         1

    accuracy                           0.80       146
   macro avg       0.33      0.30      0.29       146
weighted avg       0.66      0.80      0.72       146


Confusion Matrix:
[[  0   0  23   0]
 [  0   1   4   0]
 [  0   1 116   0]
 [  0   0   1   0]]

Gradient Boosting Model Evaluation:
Accuracy: 0.8014

Classification Report:
              precision    recall  f1-score   support

           0       0.46      0.26      0.33        23
           1       0.50      0.20      0.29         5
           2       0.85      0.94      0.89       117
           3       0.00      0.00      0.00         1

    accuracy                           0.80       146
   macro avg       0.45      0.35      0.38       146
weighted avg       0.77      0.80      0.78       146


Confusion Matrix:
[[  6   1  16   0]
 [  1   1   3   0]
 [  6   0 110   1]
 [  0   0   1   0]]
XGBoost Classifier - Accuracy: 0.8014

Classification Report:
              precision    recall  f1-score   support

           0       0.41      0.30      0.35        23
           1       0.50      0.20      0.29         5
           2       0.86      0.93      0.89       117
           3       0.00      0.00      0.00         1

    accuracy                           0.80       146
   macro avg       0.44      0.36      0.38       146
weighted avg       0.77      0.80      0.78       146
```

```
Confusion Matrix:
[[  7   1  15   0]
 [  1   1   3   0]
 [  8   0 109   0]
 [  1   0   0   0]]
```

[52]:
```
results = pd.DataFrame({
    'Model':['Decision Tree', 'Support Vector Machine', 'Random Forest',␣
 ↪'Gradient Boosting', 'XGBoost'],
    'Score':[accuracy_dt, accuracy_svm, accuracy_rf, accuracy_gb,accuracy_xgb]})
result_df = results.sort_values(by = 'Score', ascending = False)
result_df = result_df.set_index('Score')
result_df
```

[52]:
```
                           Model
Score
0.828767           Random Forest
0.801370  Support Vector Machine
0.801370       Gradient Boosting
0.801370                 XGBoost
0.753425           Decision Tree
```

Based on the table, it shows the accuracy scores for five different machine learning models: Random Forest, XGBoost, Support Vector Machine (SVM), Gradient Boosting, and Decision Tree. Among the models, we know that the highest accuracy is Random Forest with an accuracy of 82.87%, while the lowest accuracy is Decision Tree with an accuracy of 75.34%. SVM and Gradient Boosting score the same accuracy about 80.14%. In general, the results showed that the Random Forest and XGBoost models exhibit the most accurate among the five models that were evalueated.

[53]:
```
from sklearn.metrics import classification_report

# Create a DataFrame for Classification Report
classification_reports = []

# SVM
report_svm = classification_report(y_class_test, y_class_pred_svm,␣
 ↪output_dict=True)
classification_reports.append({'Model': 'SVM', **report_svm['weighted avg']})

# Decision Tree
report_dt = classification_report(y_class_test, y_class_pred_dt,␣
 ↪output_dict=True)
classification_reports.append({'Model': 'Decision Tree', **report_dt['weighted␣
 ↪avg']})

# Random Forest
```

```
report_rf = classification_report(y_class_test, y_class_pred_rf,␣
    ↪output_dict=True)
classification_reports.append({'Model': 'Random Forest', **report_rf['weighted␣
    ↪avg']})

# Gradient Boosting
report_gb = classification_report(y_class_test, y_class_pred_gb,␣
    ↪output_dict=True)
classification_reports.append({'Model': 'Gradient Boosting',␣
    ↪**report_gb['weighted avg']})

# XGBoost
report_xgb = classification_report(y_class_test, y_class_pred_xgb,␣
    ↪output_dict=True)
classification_reports.append({'Model': 'XGBoost', **report_xgb['weighted␣
    ↪avg']})


# Create DataFrame
classification_df = pd.DataFrame(classification_reports)
classification_df = classification_df.set_index('Model')

# Sort DataFrame by Score in descending order
classification_df = classification_df.sort_values(by='f1-score',␣
    ↪ascending=False)

# Print the DataFrame
print(classification_df)
```

```
                   precision    recall  f1-score  support
Model
Random Forest       0.805479  0.828767  0.793750    146.0
XGBoost             0.769780  0.801370  0.780900    146.0
Gradient Boosting   0.767914  0.801370  0.776067    146.0
Decision Tree       0.759315  0.753425  0.754077    146.0
SVM                 0.662671  0.801370  0.722114    146.0
```

Precision is concerned with positive forecast accuracy, recall is concerned with recording every true instance of success, and the F1-score achieves equilibrium between recall and accuracy. Overall, Random Forest got the highest precision, recall, and F1-score among the five models listed, with 0.81, 0.83, and 0.79, respectively. Meanwhile, SVM has the lowest precision and F1-score with 0.66 and 0.72, respectively.

9.0 ROC Curve

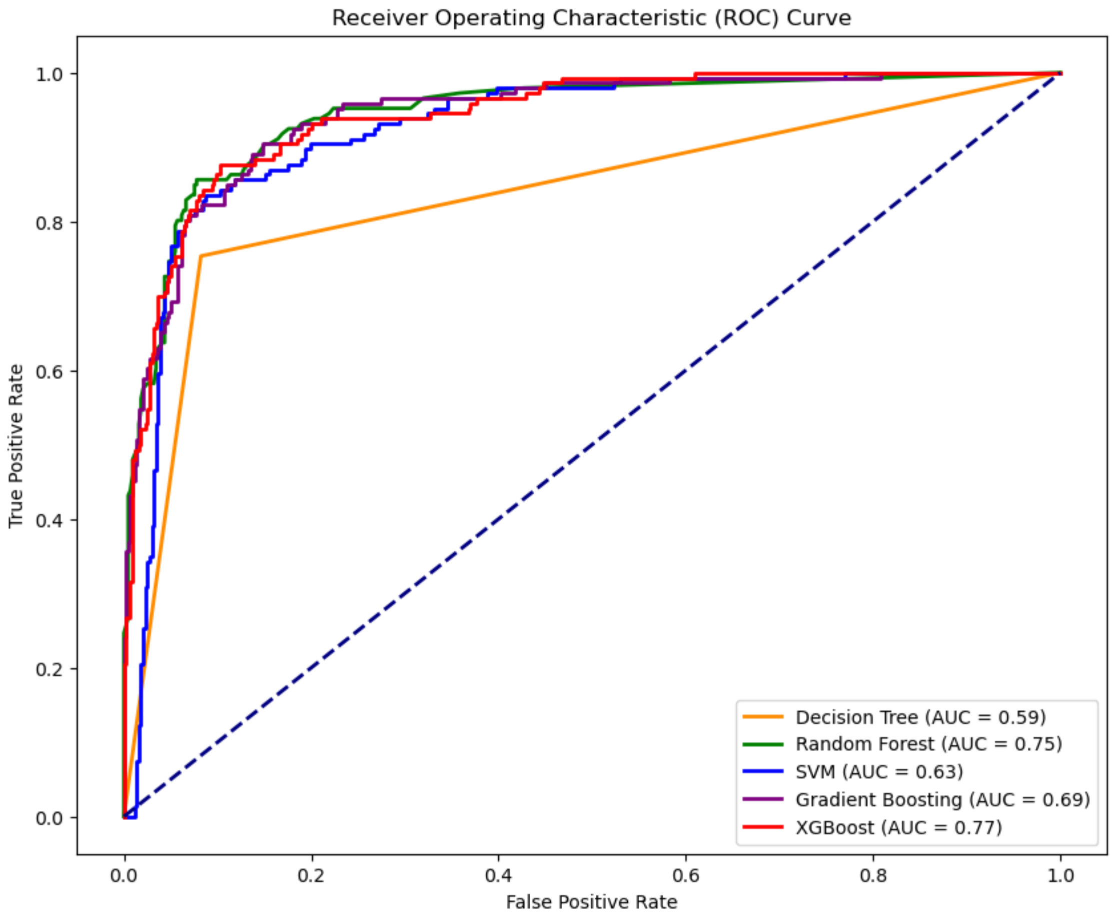
[54]:
```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

```
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import roc_curve, auc, roc_auc_score
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

# Binarize the labels
y_test_bin = label_binarize(y_class_test, classes=dt_classifier.classes_)

# Decision Tree
dt_classifier = DecisionTreeClassifier(random_state=42)
dt_classifier.fit(X_train, y_class_train)
y_dt_probs = dt_classifier.predict_proba(X_test)
roc_auc_dt = roc_auc_score(y_test_bin, y_dt_probs, multi_class='ovr')
fpr_dt, tpr_dt, thresholds_dt = roc_curve(y_test_bin.ravel(), y_dt_probs.
 ↪ravel())

# Random Forest
rf_classifier = RandomForestClassifier(random_state=42)
rf_classifier.fit(X_train, y_class_train)
y_rf_probs = rf_classifier.predict_proba(X_test)
roc_auc_rf = roc_auc_score(y_test_bin, y_rf_probs, multi_class='ovr')
fpr_rf, tpr_rf, thresholds_rf = roc_curve(y_test_bin.ravel(), y_rf_probs.
 ↪ravel())

# Support Vector Machine (SVM)
svm_classifier = SVC(probability=True, random_state=42)
svm_classifier.fit(X_train, y_class_train)
y_svm_probs = svm_classifier.predict_proba(X_test)
roc_auc_svm = roc_auc_score(y_test_bin, y_svm_probs, multi_class='ovr')
fpr_svm, tpr_svm, thresholds_svm = roc_curve(y_test_bin.ravel(), y_svm_probs.
 ↪ravel())

# Gradient Boosting
gb_classifier = GradientBoostingClassifier(random_state=42)
gb_classifier.fit(X_train, y_class_train)
y_gb_probs = gb_classifier.predict_proba(X_test)
roc_auc_gb = roc_auc_score(y_test_bin, y_gb_probs, multi_class='ovr')
fpr_gb, tpr_gb, thresholds_gb = roc_curve(y_test_bin.ravel(), y_gb_probs.
 ↪ravel())

# XGBoost
xgb_classifier = XGBClassifier(random_state=42)
xgb_classifier.fit(X_train, y_class_train)
y_xgb_probs = xgb_classifier.predict_proba(X_test)
roc_auc_xgb = roc_auc_score(y_test_bin, y_xgb_probs, multi_class='ovr')
```

```python
fpr_xgb, tpr_xgb, thresholds_xgb = roc_curve(y_test_bin.ravel(), y_xgb_probs.
    ↪ravel())

# Create a dictionary to store AUC values
auc_results = {
    'Decision Tree': roc_auc_dt,
    'Random Forest': roc_auc_rf,
    'SVM': roc_auc_svm,
    'Gradient Boosting': roc_auc_gb,
    'XGBoost': roc_auc_xgb
}

# Create DataFrame
auc_df = pd.DataFrame(list(auc_results.items()), columns=['Model', 'AUC'])
auc_df = auc_df.set_index('Model')

# Sort DataFrame by AUC in descending order
auc_df = auc_df.sort_values(by='AUC', ascending=False)

# Print the DataFrame
print(auc_df)

# Plot ROC curves
plt.figure(figsize=(10, 8))
plt.plot(fpr_dt, tpr_dt, color='darkorange', lw=2, label=f'Decision Tree (AUC =␣
    ↪{roc_auc_dt:.2f})')
plt.plot(fpr_rf, tpr_rf, color='green', lw=2, label=f'Random Forest (AUC =␣
    ↪{roc_auc_rf:.2f})')
plt.plot(fpr_svm, tpr_svm, color='blue', lw=2, label=f'SVM (AUC = {roc_auc_svm:.
    ↪2f})')
plt.plot(fpr_gb, tpr_gb, color='purple', lw=2, label=f'Gradient Boosting (AUC =␣
    ↪{roc_auc_gb:.2f})')
plt.plot(fpr_xgb, tpr_xgb, color='red', lw=2, label=f'XGBoost (AUC =␣
    ↪{roc_auc_xgb:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
                        AUC
Model
XGBoost            0.767977
Random Forest      0.752685
Gradient Boosting  0.688898
SVM                0.626958
```

```
Decision Tree      0.587532
```



Based on the graph above it showed the performance of Receiver Operating Characteristics (ROC). XGBoost has the highest Area Under the Curve (AUC) value at 0.77, indicating the most outstanding performance. At the same time, the lowest AUC is the decision tree, with only 0.59. The AUC values quantify the ability of each model to differentiate between classes. A model's performance improves as the AUC value increases. In general, the ROC curve shows that XGBoost outperforms Random Forest and Gradient Boosting as the most effective model for this task. SVM and Decision Tree exhibit comparatively lower accuracy in differentiating true positives from false positives.

10.0 MSE & RMSE

[55]:
```
# Create a DataFrame for MSE and RMSE
mse_rmse_results = []

# SVM
mse_svm = mean_squared_error(y_class_test, y_class_pred_svm)
rmse_svm = sqrt(mse_svm)
mse_rmse_results.append({'Model': 'SVM', 'MSE': mse_svm, 'RMSE': rmse_svm})
```

```
# Decision Tree
mse_dt = mean_squared_error(y_class_test, y_class_pred_dt)
rmse_dt = sqrt(mse_dt)
mse_rmse_results.append({'Model': 'Decision Tree', 'MSE': mse_dt, 'RMSE':␣
↪rmse_dt})

# Random Forest
mse_rf = mean_squared_error(y_class_test, y_class_pred_rf)
rmse_rf = sqrt(mse_rf)
mse_rmse_results.append({'Model': 'Random Forest', 'MSE': mse_rf, 'RMSE':␣
↪rmse_rf})

# Gradient Boosting
mse_gb = mean_squared_error(y_class_test, y_class_pred_gb)
rmse_gb = sqrt(mse_gb)
mse_rmse_results.append({'Model': 'Gradient Boosting', 'MSE': mse_gb, 'RMSE':␣
↪rmse_gb})

# XGBoost
mse_xgb = mean_squared_error(y_class_test, y_class_pred_xgb)
rmse_xgb = sqrt(mse_xgb)
mse_rmse_results.append({'Model': 'XGBoost', 'MSE': mse_xgb, 'RMSE': rmse_xgb})

# Create DataFrame
mse_rmse_df = pd.DataFrame(mse_rmse_results)
mse_rmse_df = mse_rmse_df.set_index('Model')

# Sort DataFrame by MSE in ascending order
mse_rmse_df = mse_rmse_df.sort_values(by='MSE', ascending=True)

# Print the DataFrame
print(mse_rmse_df)
```

```
                        MSE      RMSE
Model
Random Forest      0.582192  0.763015
Gradient Boosting  0.650685  0.806650
SVM                0.671233  0.819288
XGBoost            0.726027  0.852072
Decision Tree      0.931507  0.965146
```

The table above shows the performance of Mean Squared Error (MSE) and Root Mean Squared Error (RMSE). Among the models provided, the Random Forest stands out as the top performer due to its much lower MSE and RMSE values, with 0.58 and 0.76, respectively. With the greatest values of MSE and RMSE, the Decision Tree is the least effective of those that were provided with the MSE is 0.93 and RMSE is 0.97. The error rates are an indication of each model's prediction accuracy with regard to the objective variable. The model's performance increases as the error rate drops.

1.5.1 need to paraphrase (version 1)

Random Forest is the highest-performing model in terms of accuracy, precision, recall, and F1 score. In addition, it showcases a competitive Area Under the Curve (AUC) score and the lowest Mean Squared Error (MSE) among the models. XGBoost exhibits similar levels of accuracy, recall, and F1-score as Random Forest, with somewhat higher values for AUC and MSE.

Gradient Boosting and Support Vector Machines (SVM) provide intermediate performance in most criteria, with SVM showing a much lower accuracy. However, the Decision Tree algorithm falls short in terms of accuracy, precision, and F1-score, but it does have a satisfactory recall. Nevertheless, its elevated Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) suggest substantial disparities from the true values.

1.5.2 need to paraphrase (version 2)

Overview of the performance of the five models: Random Forest, Gradient Boosting, SVM, XGBoost, and Decision Tree.

Random Forest: This model exhibits superior performance across the majority of measures. It demonstrates the utmost accuracy, recall, and F1 score. Additionally, it has the most minimal Mean Squared inaccuracy (MSE) and Root Mean Squared Error (RMSE) values, signifying the smallest degree of inaccuracy in its forecasts. The model's accuracy is the greatest among all other models.

Gradient Boosting: This model exhibits a favorable equilibrium between accuracy, recall, and F1-score. The accuracy of this model is marginally inferior to that of Random Forest and XGBoost. The AUC value of the model is intermediate between XGBoost and SVM but superior to that of the Decision Tree. The Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) of the model are greater than those of the Random Forest and XGBoost models but less than those of the SVM and Decision Tree models.

SVM: This model has the lowest accuracy and F1 score compared to the other models. Its recall is equivalent to that of Gradient Boosting. The accuracy of this model is intermediate, falling below that of Random Forest and XGBoost while above that of Decision Tree. The AUC value of this model is inferior to all other models, with the exception of the Decision Tree model. The Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) of the model are greater than those of the Random Forest and XGBoost models but less than those of the Decision Tree model.

XGBoost: This model has the largest Area Under the Curve (AUC) value, which signifies its superior ability to differentiate between different classes accurately. The accuracy of this model is somewhat lower than that of Random Forest but greater than all other models. The accuracy, recall, and F1-score of this model are inferior to those of Random Forest but superior to those of SVM and Decision Tree. The Mean Squared Error (MSE) and Root Mean Squared Error (RMSE) of this model are greater than those of the Random Forest model but less than those of the SVM and Decision Tree models.

Decision Tree: This model has the worst performance across all measures. The model has the lowest accuracy, AUC value, and the highest MSE and RMSE values, indicating the greatest degree of inaccuracy in its predictions.

Random Forest and XGBoost consistently outperform other models across several criteria, making them the top-performing models. On the other hand, the Decision Tree is the least effective model. The selection of a model may have a substantial influence on the performance of the work, so it is essential to choose the model that most effectively aligns with the data and the unique demands of the task. It is crucial to take into account the compromises between various indicators when assessing the performance of a model. For instance, a model that exhibits high accuracy does not always imply that it will have a high F1 score or AUC value. Hence, it is essential to take into account all relevant measures while assessing the performance of a model.

[56]:
```python
import joblib

# Assuming you have trained models: dt_classifier, rf_classifier,␣
↪svm_classifier, gb_classifier, xgb_classifier

# Save Decision Tree model
joblib.dump(dt_classifier, 'decision_tree_model.pkl')

# Save Random Forest model
joblib.dump(rf_classifier, 'random_forest_model.pkl')

# Save SVM model
joblib.dump(svm_classifier, 'svm_model.pkl')

# Save Gradient Boosting model
joblib.dump(gb_classifier, 'gradient_boosting_model.pkl')

# Save XGBoost model
joblib.dump(xgb_classifier, 'xgboost_model.pkl')
```

[56]:
```
['xgboost_model.pkl']
```

[57]:
```python
# Load Decision Tree model
loaded_dt_model = joblib.load('decision_tree_model.pkl')

# Load Random Forest model
loaded_rf_model = joblib.load('random_forest_model.pkl')

# Load SVM model
loaded_svm_model = joblib.load('svm_model.pkl')

# Load Gradient Boosting model
loaded_gb_model = joblib.load('gradient_boosting_model.pkl')

# Load XGBoost model
loaded_xgb_model = joblib.load('xgboost_model.pkl')
```

[ ]:
```

```