WATER QUALITY ANALYSIS

Phase 4 : Development part 2

Objective:

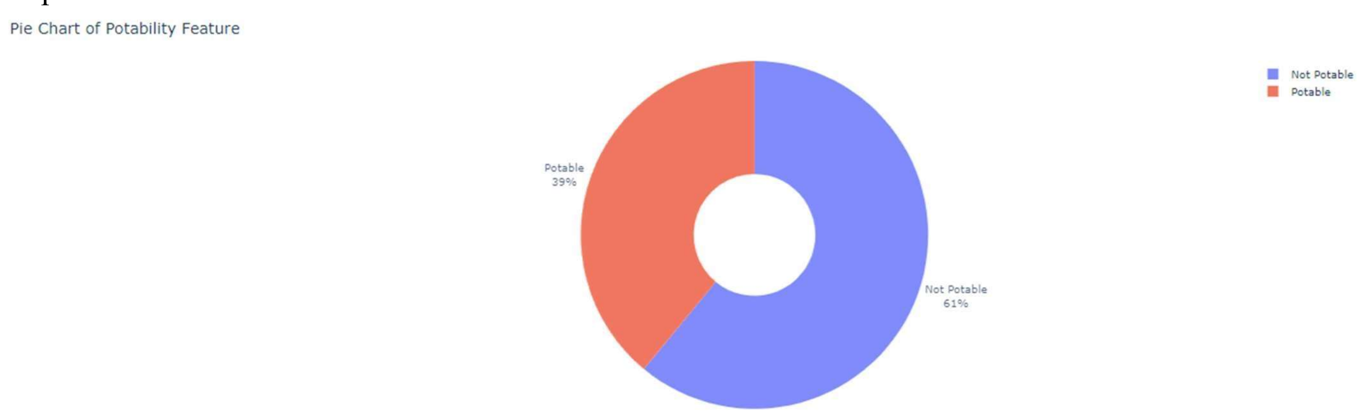
The objective of water quality analysis using IBM Cognos is to leverage advanced data analytics and reporting capabilities to assess and monitor the chemical, physical, and biological parameters of water sources. IBM Cognos enables organizations to collect, process, and visualize water quality data to ensure compliance with environmental regulations, identify contamination sources, and make informed decisions for water resource management. This analysis helps in safeguarding public health, preserving ecosystems, and optimizing water treatment processes, ultimately promoting sustainable and safe water supplies for communities and industries.

Data visualization:

Dependent Variable Analysis Program:

```
d = pd.DataFrame(df["Potability"].value_counts())
fig = px.pie(d, values = "Potability", names = ["Not Potable", "Potable"], hole = 0.35, opacity = 0.8,
            labels = {"label" :"Potability","Potability":"Number of Samples"})
fig.update_layout(title = dict(text = "Pie Chart of Potability Feature"))
fig.update_traces(textposition = "outside", textinfo = "percent+label")
fig.show()
```

Output:



Correlation Between Features

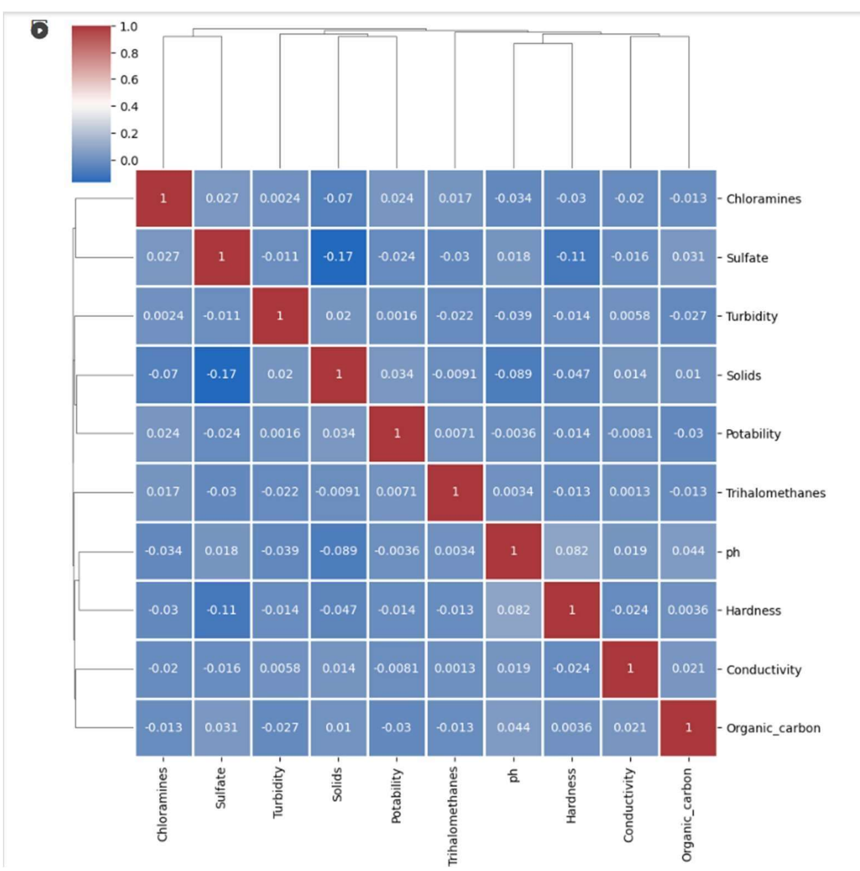
Program:

```
[ ] df.corr()
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
ph	1.000000	0.082096	-0.089288	-0.034350	0.018203	0.018614	0.043503	0.003354	-0.039057	-0.003556
Hardness	0.082096	1.000000	-0.046899	-0.030054	-0.106923	-0.023915	0.003610	-0.013013	-0.014449	-0.013837
Solids	-0.089288	-0.046899	1.000000	-0.070148	-0.171804	0.013831	0.010242	-0.009143	0.019546	0.033743
Chloramines	-0.034350	-0.030054	-0.070148	1.000000	0.027244	-0.020486	-0.012653	0.017084	0.002363	0.023779
Sulfate	0.018203	-0.106923	-0.171804	0.027244	1.000000	-0.016121	0.030831	-0.030274	-0.011187	-0.023577
Conductivity	0.018614	-0.023915	0.013831	-0.020486	-0.016121	1.000000	0.020966	0.001285	0.005798	-0.008128
Organic_carbon	0.043503	0.003610	0.010242	-0.012653	0.030831	0.020966	1.000000	-0.013274	-0.027308	-0.030001
Trihalomethanes	0.003354	-0.013013	-0.009143	0.017084	-0.030274	0.001285	-0.013274	1.000000	-0.022145	0.007130
Turbidity	-0.039057	-0.014449	0.019546	0.002363	-0.011187	0.005798	-0.027308	-0.022145	1.000000	0.001581
Potability	-0.003556	-0.013837	0.033743	0.023779	-0.023577	-0.008128	-0.030001	0.007130	0.001581	1.000000

```
sns.clustermap(df.corr(), cmap = "vlag", dendrogram_ratio = (0.1, 0.2), annot = True, linewidths = .8, figsize = (9,10))
plt.show()
```

Output:

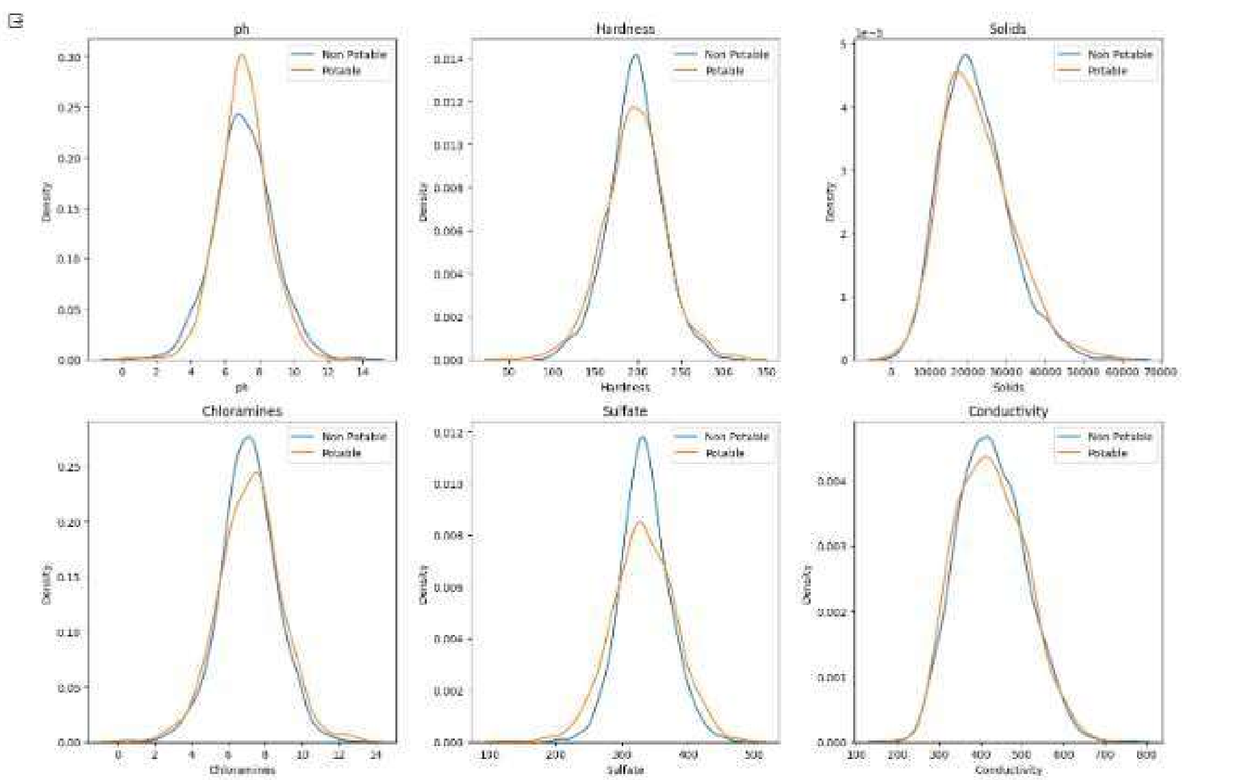


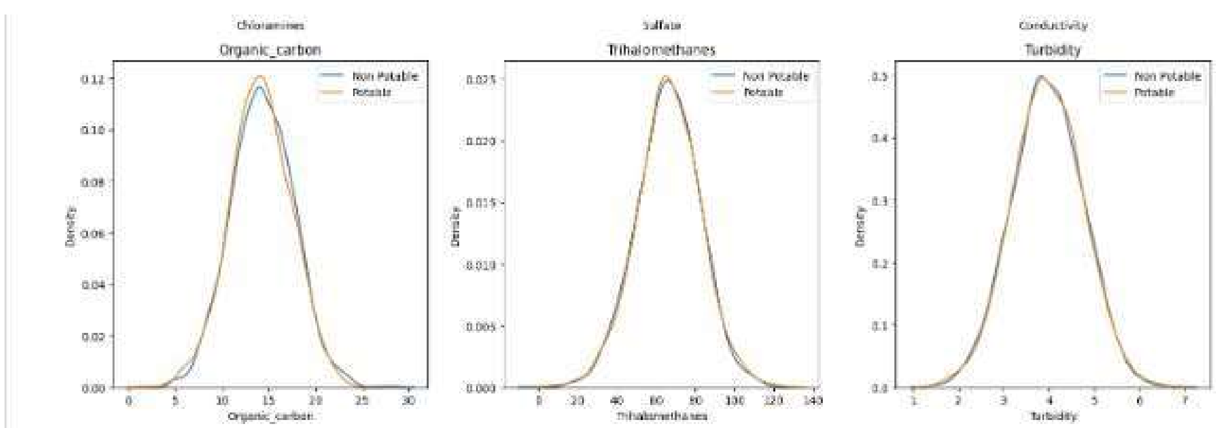
Distribution of Features:

Program:

```python
non_potable = df.query("Potability == 0")
potable = df.query("Potability == 1")

plt.figure(figsize = (15,15))
for ax, col in enumerate(df.columns[:9]):
    plt.subplot(3,3, ax + 1)
    plt.title(col)
    sns.kdeplot(x = non_potable[col], label = "Non Potable")
    sns.kdeplot(x = potable[col], label = "Potable")
    plt.legend()
plt.tight_layout()
```

Output:

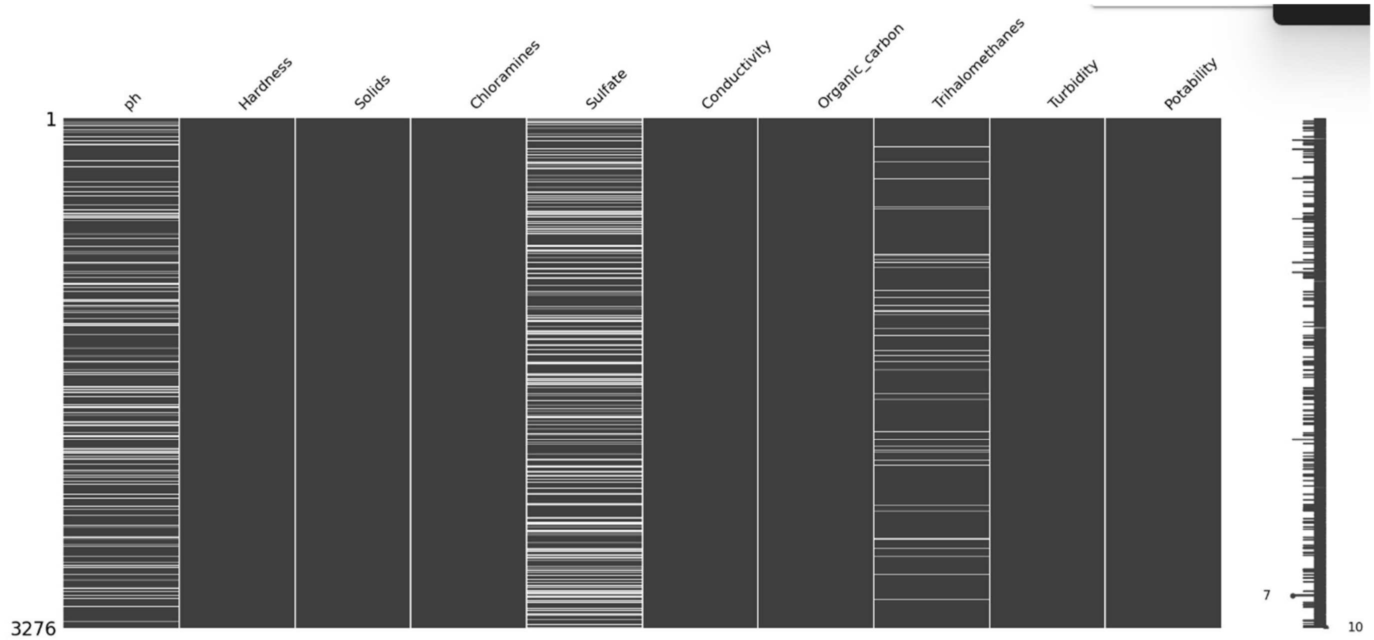




Preprocessing: Missing Value Problem:

```
msno.matrix(df)
plt.show()
```

Output:




```
df.isnull().sum()
```

```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

```
[ ] df["ph"].fillna(value = df["ph"].mean(), inplace = True)
    df["Sulfate"].fillna(value = df["Sulfate"].mean(), inplace = True)
    df["Trihalomethanes"].fillna(value = df["Trihalomethanes"].mean(), inplace = True)
```

```
df.isnull().sum()
```

```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
dtype: int64
```

Preprocessing: Train-Test Split and Normalization:

Program:

```
[ ] X = df.drop("Potability", axis = 1).values
    y = df["Potability"].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 3)
print("X_train",X_train.shape)
print("X_test",X_test.shape)
print("y_train",y_train.shape)
print("y_test",y_test.shape)
```

```
X_train (2293, 9)
X_test (983, 9)
y_train (2293,)
y_test (983,)
```

```
[ ] x_train_max = np.max(X_train)
    x_train_min = np.min(X_train)
    X_train = (X_train - x_train_min)/(x_train_max-x_train_min)
    X_test = (X_test - x_train_min)/(x_train_max-x_train_min)
```

Modelling:

Split the data and standardizing them!

Program:

[13]:

```
X = data.drop('Potability',axis=1).values
y = data['Potability'].values
```

[14]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y , test_size=0.3, random_state=101)
```

[15]:

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

# This data is imbalanced that we have more Potability -0 than 1. We will oversample in the minority class
smt = SMOTE()
X_train, y_train = smt.fit_resample(X_train, y_train)
```

We will create functions to look at AUC graph, confusion matrix and test value score to determine whether this model is valid,

[16]:

```
from sklearn import metrics

# Creating AUC plot

def model_graphs(model, model_name):

    y_pred_prob = model.predict_proba(X_test)[::,1]
    fpr, tpr, _ = metrics.roc_curve(y_test, y_pred_prob)
    auc = metrics.roc_auc_score(y_test, y_pred_prob)
    plt.plot(fpr,tpr,label= model_name +" auc="+str(auc))
    plt.legend(loc=4)
    plt.show()
```

[17]:

```
# Create confusion matrix to check accuracy, F1 score, and other

def confusion_matrix_graphs(y_pred):

    sns.heatmap(pd.DataFrame(confusion_matrix(y_test, y_pred)), annot=True, cmap="YlGnBu" ,fmt='g')
    ax.xaxis.set_label_position("top")
    plt.tight_layout()
    plt.title('Confusion matrix', y=1.1)
    plt.ylabel('Actual label')
    plt.xlabel('Predicted label')
    plt.show()
```

+ Code + Markdown

```python
# 5 folds validation and check the means accuracy score

def test_val_score(model):
    model_cross_val_score = cross_val_score(model, X_test, y_test, scoring='accuracy', cv = 5).mean()

    print("============================================================")

    print("The 5 fold cross value score is {:.2f}". format(model_cross_val_score))

    print("============================================================")
```

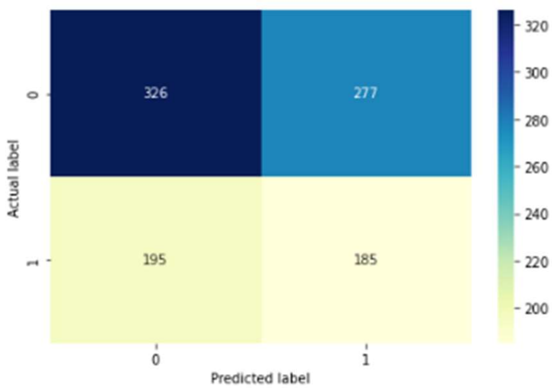
Logistric Regression:

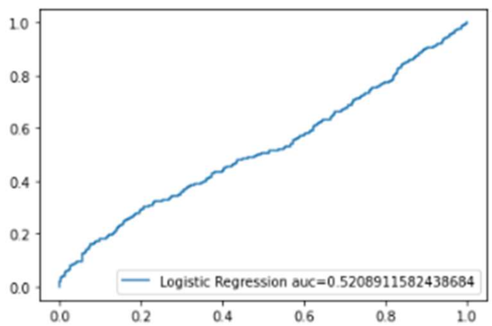
```python
from sklearn.model_selection import cross_val_score

lr = LogisticRegression()
lr.fit(X_train, y_train)
y_lr_pred = lr.predict(X_test)

test_val_score(lr)


print(classification_report(y_lr_pred, y_test))

confusion_matrix_graphs(y_lr_pred)
model_graphs(lr, "Logistic Regression")
```

Output:

```
============================================================
The 5 fold cross value score is 0.61
============================================================
              precision    recall  f1-score   support

           0       0.54      0.63      0.58       521
           1       0.49      0.40      0.44       462

    accuracy                           0.52       983
   macro avg       0.51      0.51      0.51       983
weighted avg       0.52      0.52      0.51       983
```

Confusion matrix





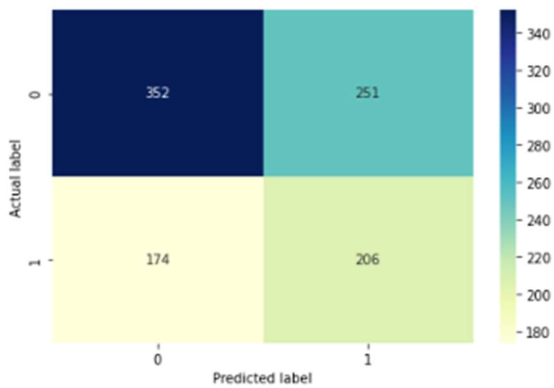
Decision Tree:
Program:

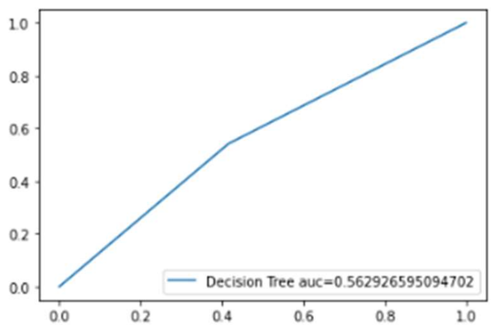
```
dt = DecisionTreeClassifier(random_state=42)
dt = dt.fit(X_train, y_train)
y_dt_pred = dt.predict(X_test)

test_val_score(dt)

print(classification_report(y_dt_pred, y_test))
confusion_matrix_graphs(y_dt_pred)
model_graphs(dt, "Decision Tree")
```

Output:

```
============================================================
The 5 fold cross value score is 0.57
============================================================
              precision    recall  f1-score   support

           0       0.58      0.67      0.62       526
           1       0.54      0.45      0.49       457

    accuracy                           0.57       983
   macro avg       0.56      0.56      0.56       983
weighted avg       0.56      0.57      0.56       983
```

Confusion matrix





+ Code + Markdown

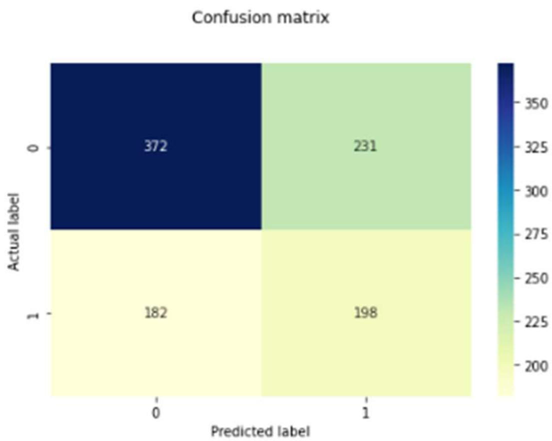
KNN:

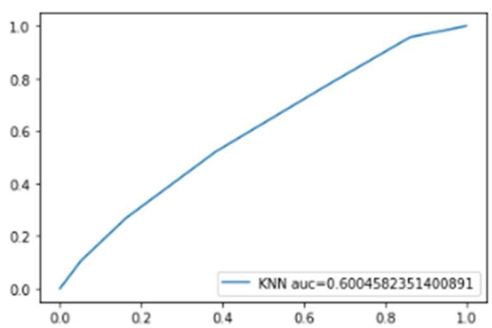
Program:

[22]:
```
KNN = KNeighborsClassifier()
KNN = KNN.fit(X_train, y_train)
y_knn_pred = KNN.predict(X_test)

test_val_score(KNN)

print(classification_report(y_knn_pred, y_test))

confusion_matrix_graphs(y_knn_pred)

model_graphs(KNN, "KNN")
```

Output:

```
==========================================================
The 5 fold cross value score is 0.61
==========================================================
              precision    recall  f1-score   support

           0       0.62      0.67      0.64       554
           1       0.52      0.46      0.49       429

    accuracy                           0.58       983
   macro avg       0.57      0.57      0.57       983
weighted avg       0.58      0.58      0.58       983
```





Naive Bayes:

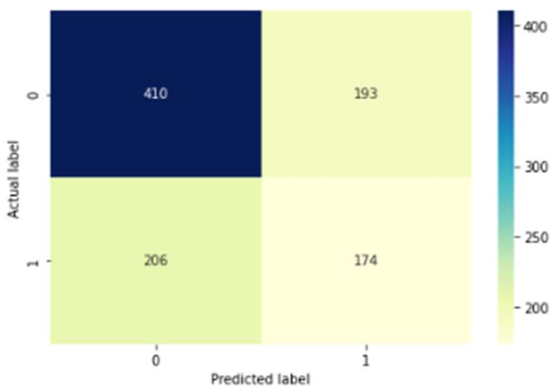
Program:

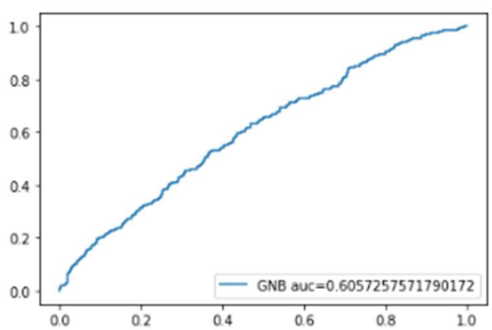
```
[23]:
GNB = GaussianNB()
GNB = GNB.fit(X_train, y_train)
y_GNB_pred = GNB.predict(X_test)

test_val_score(GNB)

print(classification_report(y_GNB_pred, y_test))

confusion_matrix_graphs(y_GNB_pred)

model_graphs(GNB, "GNB")
```

Output:

```
==========================================================
The 5 fold cross value score is 0.63
==========================================================
              precision    recall  f1-score   support

           0       0.68      0.67      0.67       616
           1       0.46      0.47      0.47       367

    accuracy                           0.59       983
   macro avg       0.57      0.57      0.57       983
weighted avg       0.60      0.59      0.60       983
```

Confusion matrix





Conclusion:

In conclusion, water analysis is a critical process for assessing the quality and safety of water resources. It involves a series of steps, from data collection and preprocessing to in-depth analysis and interpretation. By rigorously examining water quality data, we can make informed decisions, safeguard public health, and protect the environment, ensuring the availability of clean and safe water for all.