**Phase 4: Development part 2**

**Project Title: Water Quality Analysis**

**Problem :**

This project involves analyzing water quality data to assess the suitability of water for specific purposes, such as drinking. The objective is to identify potential issues or deviations from feature engineering,Modelling and Evaluating the dataset.

**Definition:**

 The ph regulatory standards and determine water potability based on various parameters. This project includes defining analysis objectives, collecting water quality data, designing relevant visualizations, and import numpy as np .

**Feature Engineering:**

**1.Data Collection and Compilation**:

* Gather water quality data, including parameters like pH, hardness, solids, chlorine, sulphate, organic carbon, conductivity, turbidity, and trihalomethanes, as well as information about whether the water is potable or not.

**2. Feature Selection:**

* Analyze the correlation between each parameter and potability. Select the relevant features that have a significant impact on water potability
* You can use correlation matrices, feature importance techniques, or domain knowledge.

**3. Data Preprocessing:**

* Handle missing values, outliers, and inconsistencies in the data.
* Normalize or scale features, if necessary, to ensure they are on the same scale.

**4. Feature Transformation:**

* Consider engineering new features that capture interactions or nonlinear relationships between parameters.

**5. Encoding:**

* If potability is a categorical variable (e.g., 0 for non-potable and 1 for potable), encode it appropriately for modeling.

**Model Training:**

**1.Data Splitting:**

* Split the dataset into training and testing sets. You can use techniques like stratified sampling to ensure class balance.

**2.Selecting a Model:**

* Choose an appropriate machine learning algorithm for binary classification. Common choices include logistic regression, random forests, support vector machines, or neural networks.
* Consider the nature of your data and the model's interpretability.

**3.Model Training:**

* Train the selected model on the training data, using the features to predict water potability.

**4.Model Tuning:**

* Optimize hyperparameters of the model to improve its performance.
* Techniques like grid search or random search can be helpful.

**Evaluation:**

**1. Model Evaluation Metrics:**

* Evaluate the model's performance using appropriate metrics, such as accuracy, precision, recall, F1-score, and the area under the ROC curve (AUC).
* For a drinking water model, it is essential to focus on both accuracy and sensitivity (recall) to ensure safe drinking water.

**2.Confusion Matrix:**

* Examine the confusion matrix to understand the model's true positives, true negatives, false positives, and false negatives.

**3.Cross-Validation:**

* Implement cross-validation to ensure the model's robustness and reduce overfitting.

**4.Interpretability:**

* Interpret the model to understand which features are most influential in determining water potability.

**5.Model Deployment:**

* If the model performs well and meets the criteria for safe drinking water, deploy it in a real-world setting for continuous monitoring and decision-making.

**6.Monitoring and Maintenance:**

* Regularly monitor the model's performance and update it as needed to account for changing water quality conditions or new data.

**7.Documentation:**

* Keep comprehensive records of the entire process, including data sources, preprocessing steps, model details, and evaluation results for transparency and reproducibility.

By following these steps, you can develop a robust and reliable model to assess the potability of water based on various water quality parameters. It's essential to ensure that the model is accurate and safe for drinking water decisions.

#Exploratory Data Analysis

Corrmat = df.corr()

plt.subplots(figsize=(7,7))

sns.heatmap(Corrmat, cmap="YlGnBu", square = True, annot=True, fmt='.2f')

plt.show()

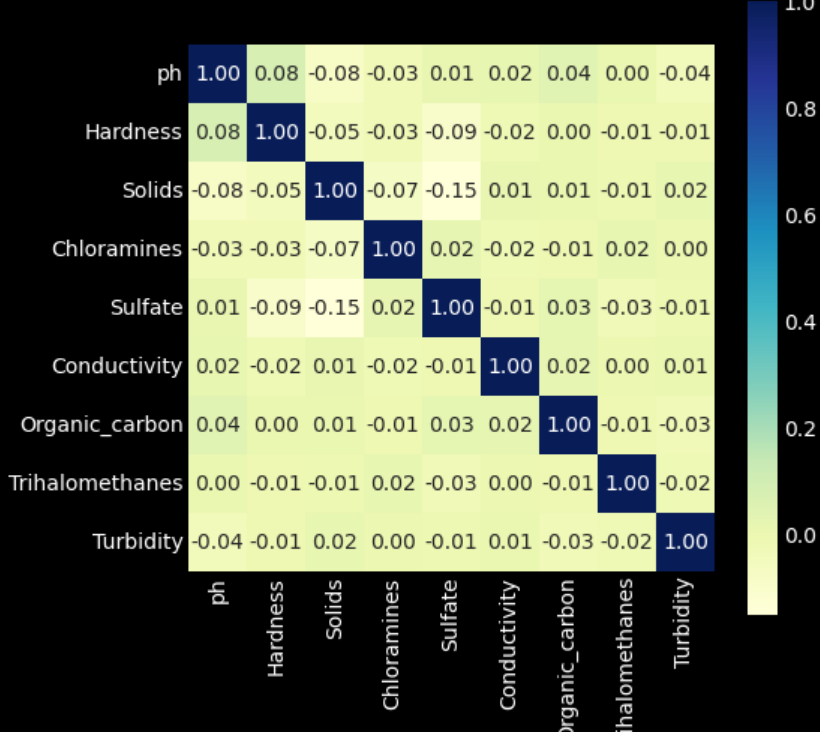
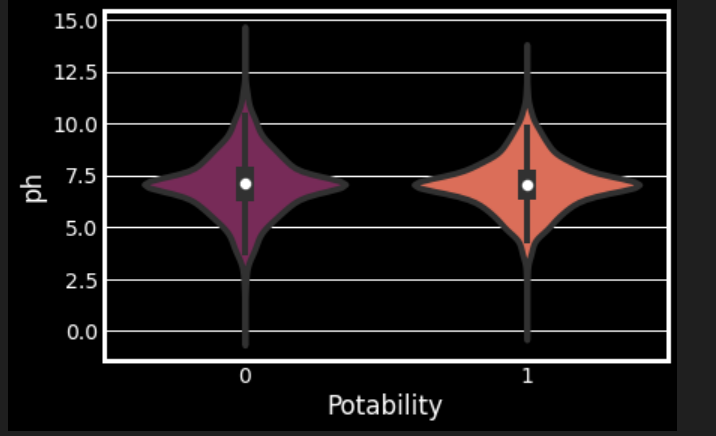


fig = ex.pie (df, names = "Potability", hole = 0.4, template = "plotly\_dark")

fig.show ()

sns.violinplot(x='Potability', y='ph', data=df, palette='rocket')



print('Boxplot and density distribution of different features by Potability\n')

fig, ax = plt.subplots(ncols=2, nrows=9, figsize=(14, 28))

features = list(df.columns.drop('Potability'))

i=0

for cols in features:

sns.kdeplot(df[cols], fill=True, alpha=0.4, hue = df.Potability,

palette=('indianred', 'steelblue'), multiple='stack', ax=ax[i,0])

sns.boxplot(data= df, y=cols, x='Potability', ax=ax[i, 1],

palette=('indianred', 'steelblue'))

ax[i,0].set\_xlabel(' ')

ax[i,1].set\_xlabel(' ')

ax[i,1].set\_ylabel(' ')

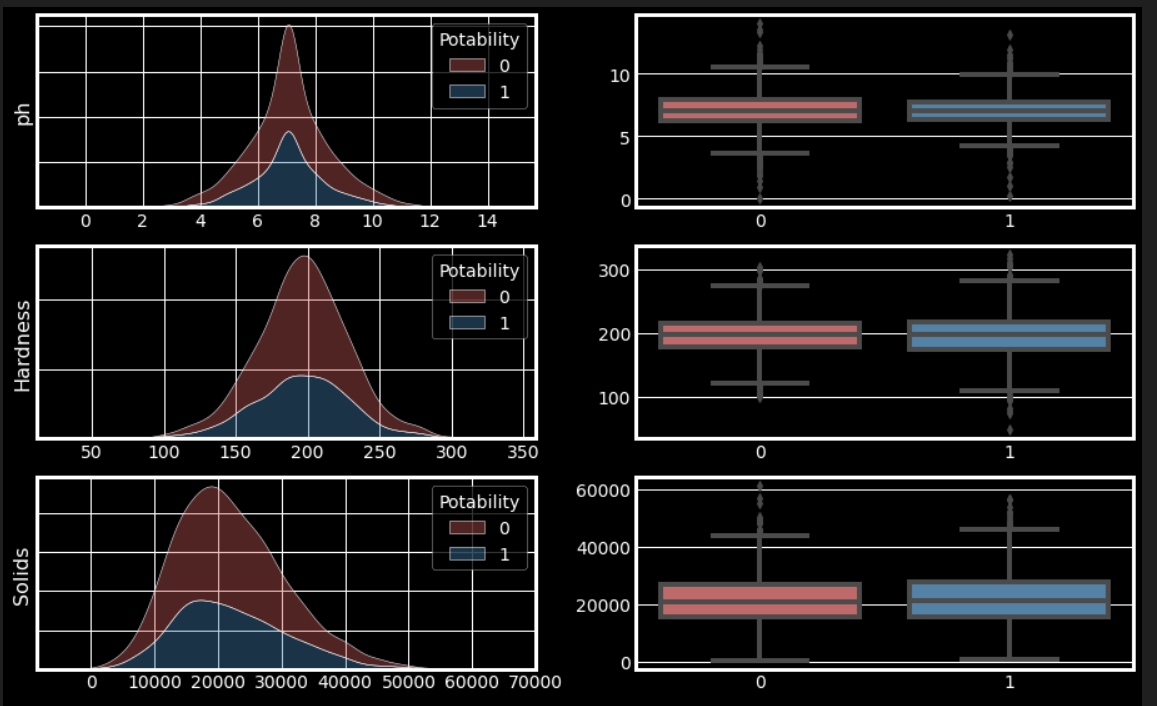
ax[i,1].xaxis.set\_tick\_params(labelsize=14)

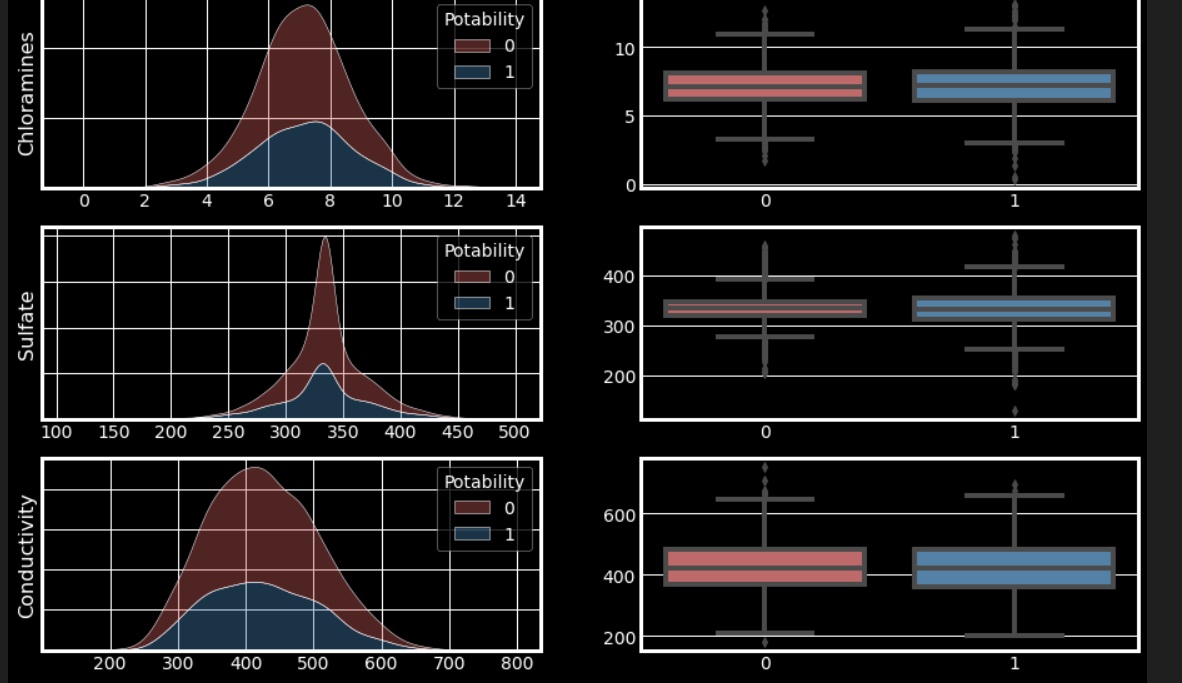
ax[i,0].tick\_params(left=False, labelleft=False)

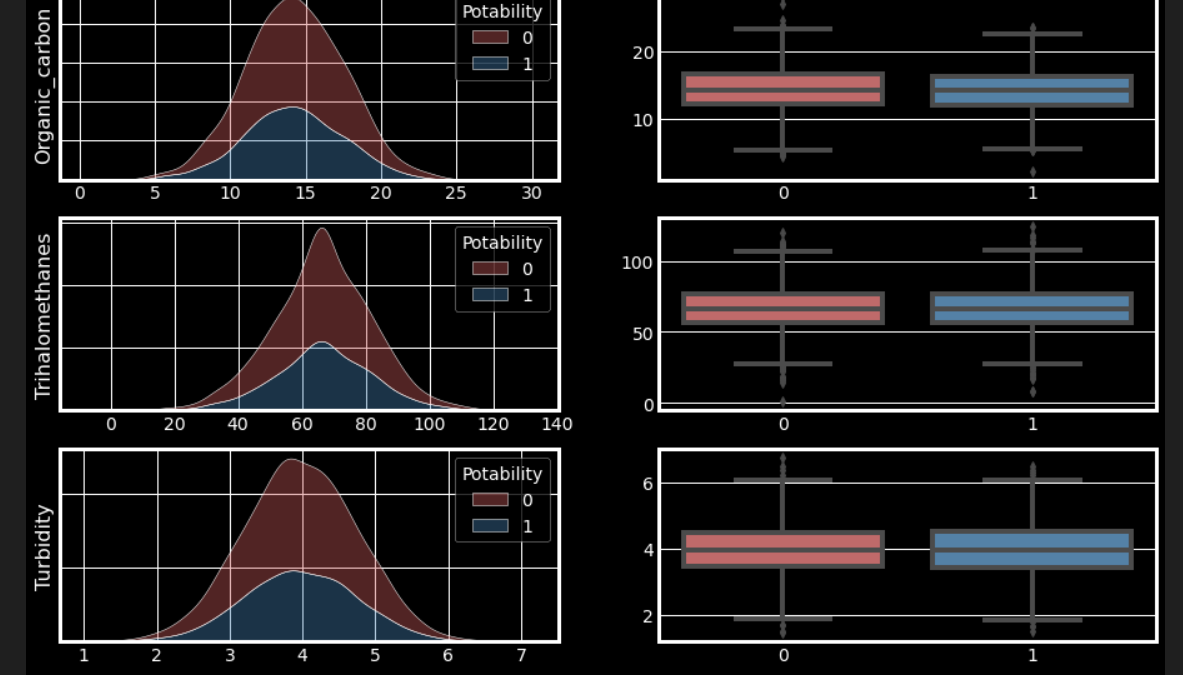
ax[i,0].set\_ylabel(cols, fontsize=16)

i=i+1

plt.show()







#SMOTE

#Preparing the Data for Modelling

X = df.drop('Potability', axis = 1).copy()

y = df['Potability'].copy()

Train-Test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y,test\_size=0.25)

# Synthetic OverSampling

print('Balancing the data by SMOTE - Oversampling of Minority level\n')

smt = SMOTE()

counter = Counter(y\_train)

print('Before SMOTE', counter)

X\_train, y\_train = smt.fit\_resample(X\_train, y\_train)

counter = Counter(y\_train)

print('\nAfter SMOTE', counter)

# Scaling

ssc = StandardScaler()

X\_train = ssc.fit\_transform(X\_train)

X\_test = ssc.transform(X\_test)

modelAccuracy = list()

## Modelling and Prediction

model = [LogisticRegression(), DecisionTreeClassifier(), GaussianNB(), RandomForestClassifier(),

svm.LinearSVC(), XGBClassifier()]

trainAccuracy = list()

testAccuracy = list()

kfold = KFold(n\_splits=10, random\_state=7, shuffle=True)

for mdl in model:

trainResult = cross\_val\_score(mdl, X\_train, y\_train, scoring='accuracy', cv=kfold)

trainAccuracy.append(trainResult.mean())

mdl.fit(X\_train, y\_train)

y\_pred = mdl.predict(X\_test)

testResult = metrics.accuracy\_score(y\_test, y\_pred)

testAccuracy.append(testResult)

print('The comparision\n')

modelScore = pd.DataFrame({'Model' : model, 'Train\_Accuracy' : trainAccuracy, 'Test\_Accuracy' : testAccuracy})

modelScore

print('Random Forest Classifier\n')

Rfc = RandomForestClassifier()

Rfc.fit(X\_train, y\_train)

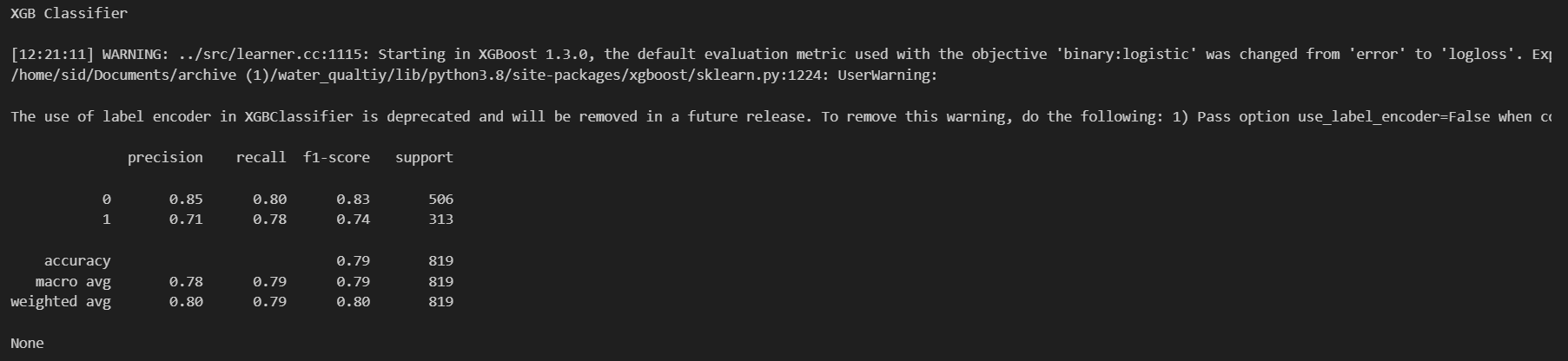
y\_Rfc = Rfc.predict(X\_test)

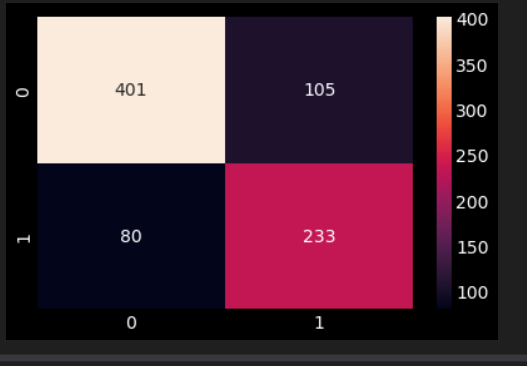
print(metrics.classification\_report(y\_test, y\_Rfc))

print(modelAccuracy.append(metrics.accuracy\_score(y\_test, y\_Rfc)))

sns.heatmap(confusion\_matrix(y\_test, y\_Rfc), annot=True, fmt='d')

plt.show()



print('XGBClassifier\n')

xgb = XGBClassifier()

xgb.fit(X\_train, y\_train)

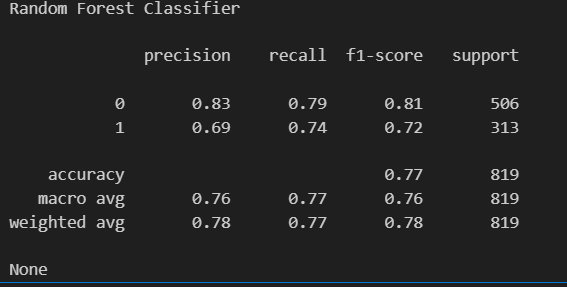
y\_xgb = xgb.predict(X\_test)

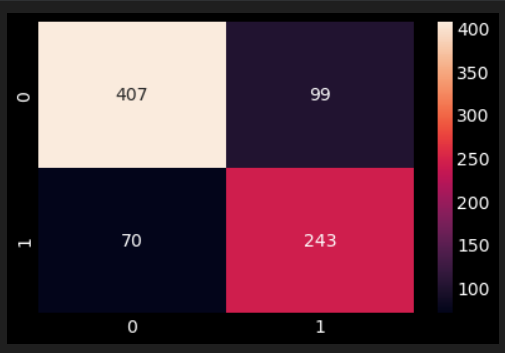
print(metrics.classification\_report(y\_test, y\_xgb))

print(modelAccuracy.append(metrics.accuracy\_score(y\_test, y\_xgb)))

sns.heatmap(confusion\_matrix(y\_test, y\_xgb), annot=True, fmt='d')

plt.show()





**Conclusion:**

* The Solid levels seem to contain some descripency since its values are on an average 40 folds more than the upper limit for safe drinking water.(Desirable limit for TDS is 500 mg/l and maximum limit is 1000 mg/l which prescribed for drinking purpose.)
* The data contains almost equal number of acidic and basic pH level water samples.
* The correlation coefficients between the features were very low.
* Random Forest and XGBoost worked the best to train the model, both gives us f1 score (Balanced with precision & recall) as around 76%.