American International University-Bangladesh (AIUB)
Department of Computer Science
Faculty of Science &Technology
(FST)Spring: 22-23

Section: [A]

PROGRAMMING IN PYTHON

Project Name: Water potability prediction using classificationmodels.

A Report submitted by

SN	Student Name	Student ID
1	ABDULLAH YOUSUF NOMAN	20-42130-1

Under the supervision of

DR. AKINUL ISLAM JONY

Associated Professor & Head (UG), Computer Science

Project Overview:

The potability of water is a critical aspect of environmental health. The objective of this project is to classify water potability data into various categories using machine learning algorithms in Python. The dataset used in this project contains various parameters like pH, dissolved oxygen, conductivity, temperature, etc., which are used to classify water quality into different categories like drinkable, non-drinkable, and suitable for industrial use. We will use five popular machine learning algorithms, namely Naive Bayes, K-Nearest Neighbors (KNN), Decision Tree, Logistic Regression, and Support Vector Machine (SVM) to classify the water quality data.Naive Bayes. We are gonna Use

1. K-Nearest Neighbors (KNN)
2. Decision Tree
3. Logistic Regression
4. Support Vector Machine (SVM)
5. Naive Bayes

We have used some important libraries like pandas, matplotlib, seaborn, scikit-learn etc. to process our dataset and make some proper visualization of the dataset. In the data cleaning process, extra columns have been removed, and missing values have been handled using forward fill and backward fill methods. To display the relationship between the features, the correlation matrix has also been presented as a heat map. We have also reduced some columns that are highly correlated to each other. The train_test_split method from scikit-learn has been used to divide the dataset into training and testing sets. The models' performance has been assessed after they have been trained on the training set and tested on the testing set. Based on the prediction accuracy of the models, which has been calculated using the scikit-learn accuracy score technique, the models' performance has been assessed. After getting all the accuracy from those models we have created a barplot for better understanding of the result.

Dataset Overview:

The dataset contains water quality metrics for 3276 different water bodies. The metrics are pH value, hardness, total dissolved solids (TDS), chloramines, sulfate, conductivity, total organic carbon (TOC), trihalomethanes (THMs), turbidity, and potability. These metrics are important for evaluating the safety of drinking water. The dataset provides information on the range and standards of each metric and whether the water is safe for human consumption.
The columns are

Pure water is not a good conductor of electric current rather's a good insulator. Increase in ions concentration enhances the electrical

Sulfate concentration in seawater is about 2,700 milligrams per liter (mg/L). It ranges from 3 to 30 mg/L in most freshwater supplies, although much higher concentrations (1000 mg/L) are found in some geographic locations.

1. pH value:

PH is an important parameter in evaluating the acid–base balance of water.

It is also the indicator of acidic or alkaline condition of water status.

WHO has recommended maximum permissible limit of pH from 6.5 to 8.5. The

current investigation ranges were 6.52–6.83 which are in the range of WHO

standards.

2. Hardness:

Hardness is mainly caused by calcium and magnesium salts. These salts are

dissolved from geologic deposits through which water travels. The length

of time water is in contact with hardness producing material helps

determine how much hardness there is in raw water. Hardness was originally

defined as the capacity of water to precipitate soap caused by Calcium and

Magnesium.

3. Solids (Total dissolved solids - TDS):

4. Chloramines:

Chlorine and chloramine are the major disinfectants used in public water

systems. Chloramines are most commonly formed when ammonia is added to

chlorine to treat drinking water. Chlorine levels up to 4 milligrams per

liter (mg/L or 4 parts per million (ppm)) are considered safe in drinking

water.

5. Sulfate:

6. Conductivity:

The turbidity of water depends on the quantity of solid matter present in

Indicates if water is safe for human consumption where 1 means Potable and 0 means Not potable.

Genet Campus (0.98 NTU) is lower than the WHO recommended value of 5.00 NTU.

conductivity of water. Generally, the amount of dissolved solids in water

determines the electrical conductivity. Electrical conductivity (EC)

actually measures the ionic process of a solution that enables it to

transmit current. According to WHO standards, EC value should not exceeded

400 µS/ cm.

7. Organic_carbon:

Total Organic Carbon (TOC) in source waters comes from decaying natural

organic matter (NOM) as well as synthetic sources. TOC is a measure of the

total amount of carbon in organic compounds in pure water. According to US

EPA < 2 mg/L as TOC in treated / drinking water, and < 4 mg/Lit in source

water which is use for treatment.

8. Trihalomethanes:

9. Turbidity:

10. Potability:

Source: https://www.kaggle.com/datasets/adityakadiwal/water-potability

1. **Importing all the necessary libraries and then loading our dataset into jupyter notebook. After that we are printing 5 rows of our dataset.**

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
```

```
In [2]: train_data = pd.read_csv("water_potability.csv")
```

```
In [3]: train_data.head(5)
```

Out[3]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0

2.Printing the shape, all the keys and a description of our dataset

```
In [4]: train_data.shape
```

```
Out[4]: (3276, 10)
```

```
In [5]: train_data.keys()
```

```
Out[5]: Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
               'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],
              dtype='object')
```

```
In [6]: train_data.describe()
```

Out[6]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.000000	3114.000000	3276.000000	3276.000000
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	14.284970	66.396293	3.966786	0.390110
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	3.308162	16.175008	0.780382	0.487849
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	0.738000	1.450000	0.000000
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414	12.065801	55.844536	3.439711	0.000000
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	14.218338	66.622485	3.955028	0.000000
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304	16.557652	77.337473	4.500320	1.000000
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.000000

3.Checking if there is any null value in our dataset columns

```
In [7]: train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   ph               2785 non-null   float64
 1   Hardness         3276 non-null   float64
 2   Solids           3276 non-null   float64
 3   Chloramines      3276 non-null   float64
 4   Sulfate          2495 non-null   float64
 5   Conductivity     3276 non-null   float64
 6   Organic_carbon   3276 non-null   float64
 7   Trihalomethanes  3114 non-null   float64
 8   Turbidity        3276 non-null   float64
 9   Potability       3276 non-null   int64  
dtypes: float64(9), int64(1)
memory usage: 256.1 KB
```

```
In [8]: train_data.isnull().sum()/len(train_data)
```

```
Out[8]: ph                 0.149878
Hardness           0.000000
Solids             0.000000
Chloramines        0.000000
Sulfate            0.238400
Conductivity       0.000000
Organic_carbon     0.000000
Trihalomethanes    0.049451
Turbidity          0.000000
Potability         0.000000
dtype: float64
```

4. As ph, Sulfate, Trihalomethanes contains null value we are replacing those null value with the mean value.

```
In [9]: cols=['ph','Sulfate','Trihalomethanes']
        train_data[cols].mean()
```

```
Out[9]: ph                   7.080795
        Sulfate            333.775777
        Trihalomethanes     66.396293
        dtype: float64
```

```
In [10]: train_data['ph'].fillna((train_data['ph'].mean()), inplace=True)
         train_data['Sulfate'].fillna((train_data['Sulfate'].mean()), inplace=True)
         train_data['Trihalomethanes'].fillna((train_data['Trihalomethanes'].mean()), inplace=True)
         train_data
```

Out[10]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	7.080795	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0
1	3.716080	129.422921	18630.057858	6.635246	333.775777	592.885359	15.180013	56.329076	4.500656	0
2	8.099124	224.236259	19909.541732	9.275884	333.775777	418.606213	16.868637	66.420093	3.055934	0
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	0
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0
...	...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	1
3272	7.808856	193.553212	17329.802160	8.061362	333.775777	392.449580	19.903225	66.396293	2.798243	1
3273	9.419510	175.762646	33155.578218	7.350233	333.775777	432.044783	11.039070	69.845400	3.298875	1
3274	5.126763	230.603758	11983.869376	6.303357	333.775777	402.883113	11.168946	77.488213	4.708658	1
3275	7.874671	195.102299	17404.177061	7.509306	333.775777	327.459760	16.140368	78.698446	2.309149	1

3276 rows × 10 columns

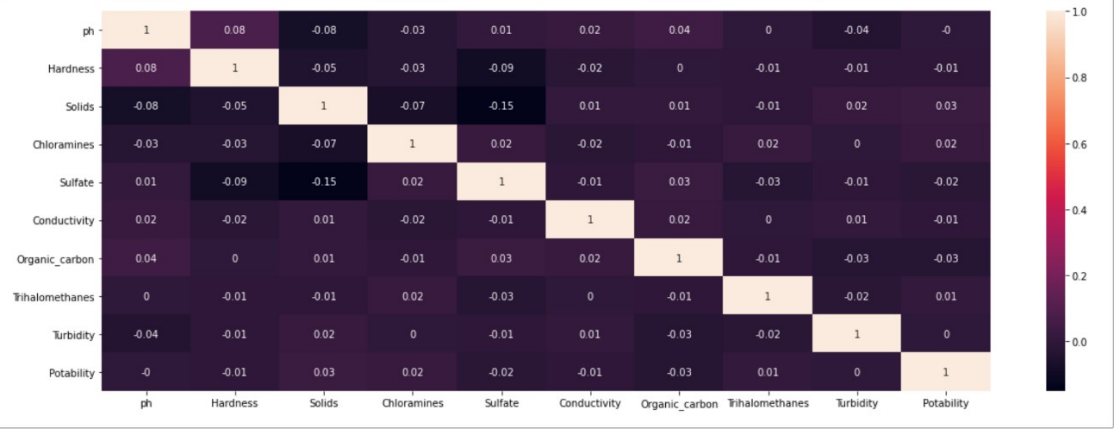
5. Again we are checking if our data contains any null value or not.

```
In [11]: train_data.isnull().sum()/len(train_data)
```

```
Out[11]: ph                0.0
         Hardness          0.0
         Solids            0.0
         Chloramines       0.0
         Sulfate           0.0
         Conductivity      0.0
         Organic_carbon    0.0
         Trihalomethanes   0.0
         Turbidity         0.0
         Potability        0.0
         dtype: float64
```

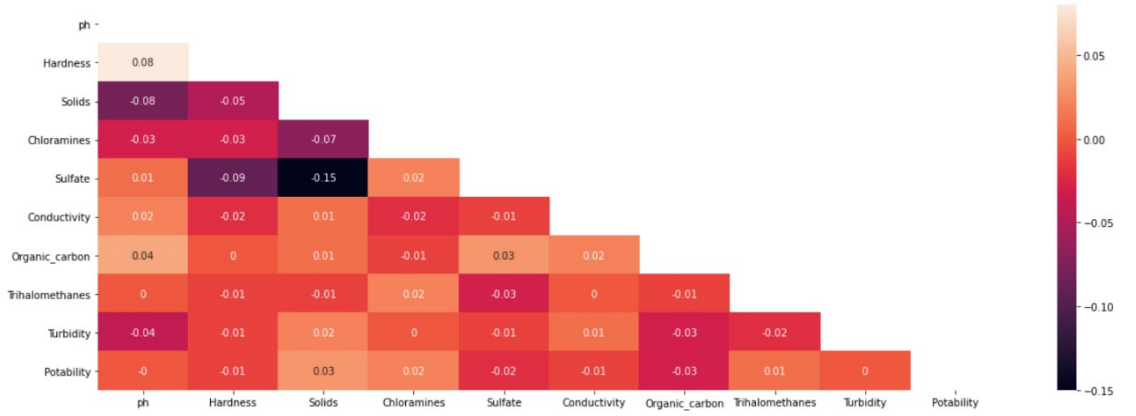
6. Building a correlation matrix for checking if any highly correlated columns are present or not.

```
In [50]: # corr() to calculate the correlation between variables
         correlation_matrix = train_data.corr().round(2)
         # changing the figure size
         plt.figure(figsize = (20, 7))
         # "annot = True" to print the values inside the square
         sns.heatmap(data=correlation_matrix, annot=True);
```



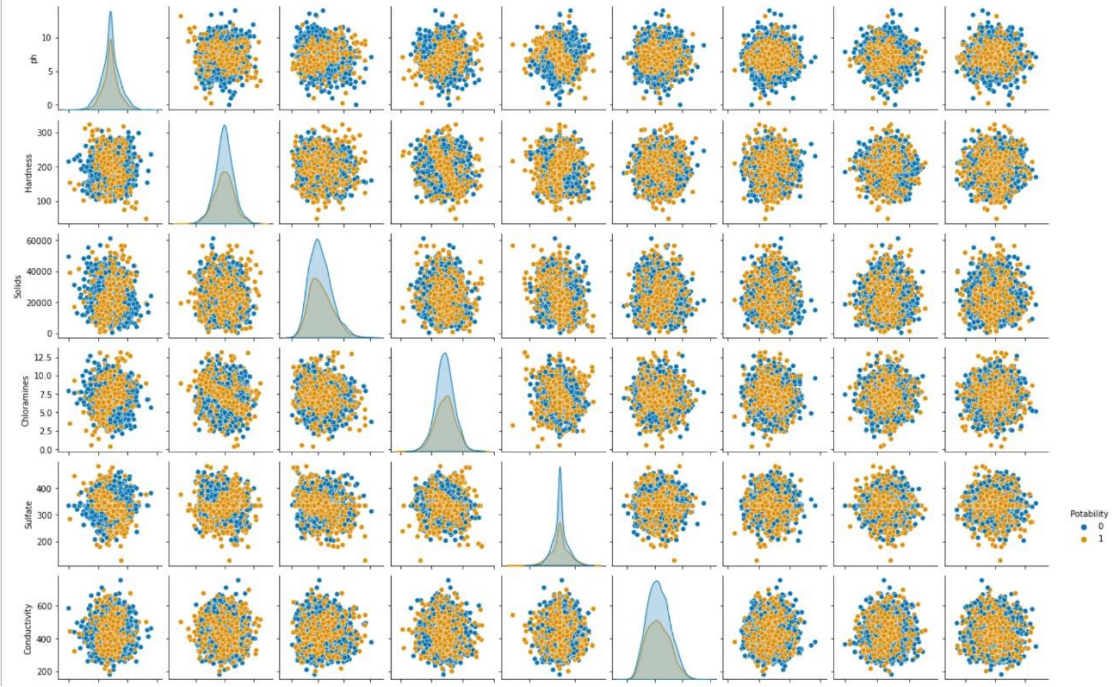
7. **As we can see there is no highly correlated columns in our dataset. So wedon't need to drop any columns.**

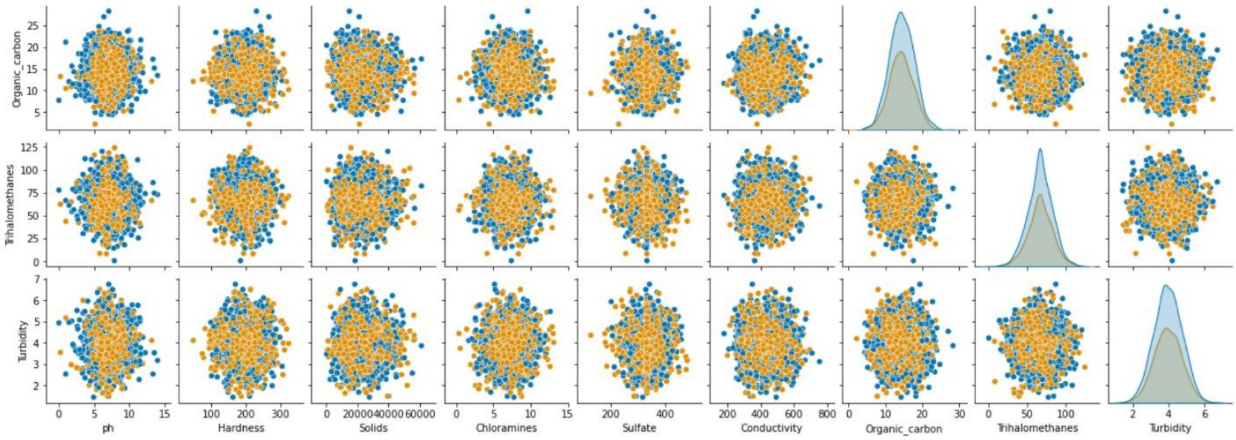
```
In [51]: # Steps to remove redundant values
         # Return a array filled with zeros
         mask = np.zeros_like(correlation_matrix)
         # Return the indices for the upper-triangle of array
         mask[np.triu_indices_from(mask)] = True
         # changing the figure size
         plt.figure(figsize = (20, 7))
         # "annot = True" to print the values inside the square
         sns.heatmap(data=correlation_matrix, annot=True, mask=mask);
```



8. **Building pairpolot for our dataset to see the data distribution as pairs**

```
In [53]: sns.pairplot(train_data, hue="Potability", height = 2, palette = 'colorblind');
```

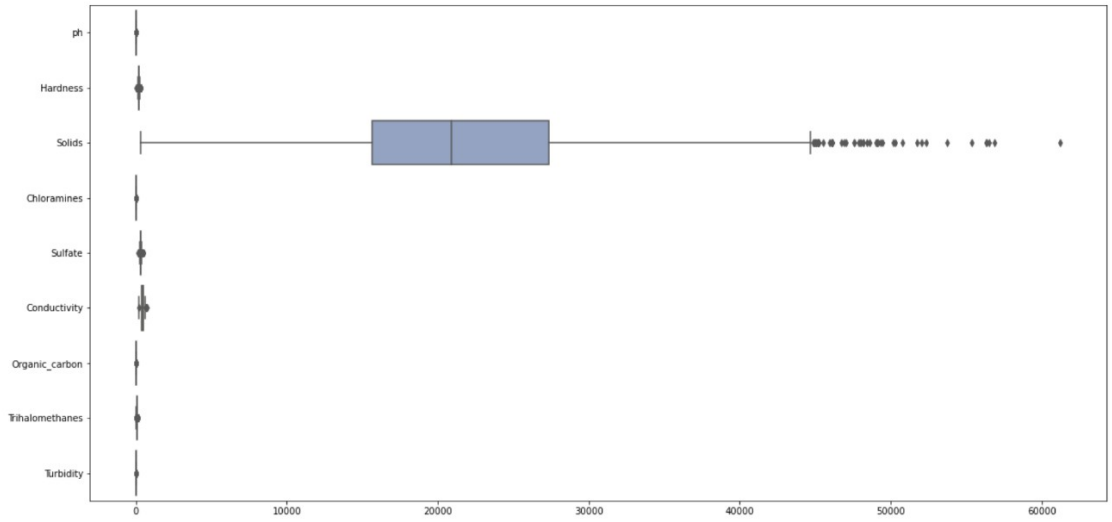




9. Building a boxplot based on our dataset to detect the outliers. As we cansee, 'solids' columns contains more outliers

```
In [78]: plt.figure(figsize=(20,10))
         sns.boxplot(data=train_data, orient="h", palette="Set2")
```

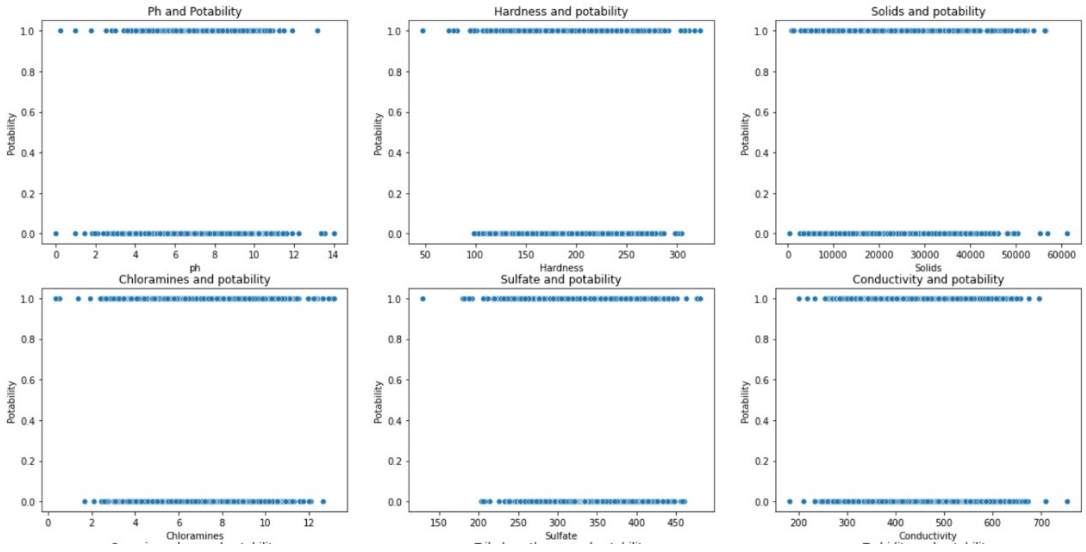
```
Out[78]: <AxesSubplot:>
```

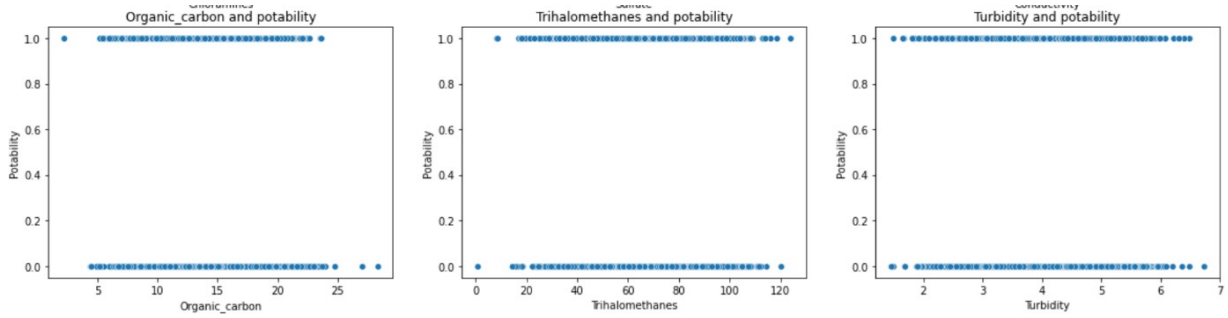


10. Building a scatter plot to see the data distribuition against our target vectorwhich is ‘Potability’

In [54]:
```
plt.figure(figsize=(20,15))

plt.subplot(3, 3, 1)
sns.scatterplot(x=train_data['ph'], y=train_data['Potability'])
plt.title("Ph and Potability")

plt.subplot(3, 3, 2)
sns.scatterplot(x=train_data['Hardness'], y=train_data['Potability'])
plt.title("Hardness and potability")

plt.subplot(3, 3, 3)
sns.scatterplot(x=train_data['Solids'], y=train_data['Potability'])
plt.title("Solids and potability")

plt.subplot(3, 3, 4)
sns.scatterplot(x=train_data['Chloramines'], y=train_data['Potability'])
plt.title("Chloramines and potability")

plt.subplot(3, 3, 5)
sns.scatterplot(x=train_data['Sulfate'], y=train_data['Potability'])
plt.title("Sulfate and potability")

plt.subplot(3, 3, 6)
sns.scatterplot(x=train_data['Conductivity'], y=train_data['Potability'])
plt.title("Conductivity and potability")

plt.subplot(3, 3, 7)
sns.scatterplot(x=train_data['Organic_carbon'], y=train_data['Potability'])
plt.title("Organic_carbon and potability")

plt.subplot(3, 3, 8)
sns.scatterplot(x=train_data['Trihalomethanes'], y=train_data['Potability'])
plt.title("Trihalomethanes and potability")

plt.subplot(3, 3, 9)
sns.scatterplot(x=train_data['Turbidity'], y=train_data['Potability'])
plt.title("Turbidity and potability")
```

Out[54]: Text(0.5, 1.0, 'Turbidity and potability')





11. Converting our target vector from numerical to catagorical.

In [62]:
```
train_data.dtypes
train_data['Potability'] = train_data['Potability'].astype('category')
train_data.loc[:,'Potability'] = train_data['Potability'].map({0: 'Not potable', 1: 'Potable'})
train_data
```

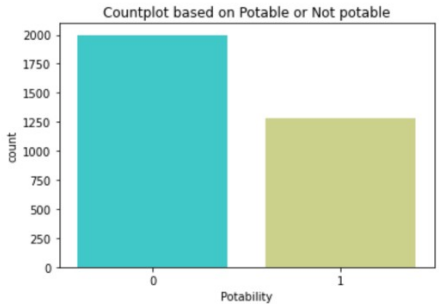
Out[62]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
0	7.080795	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	Not potable
1	3.716080	129.422921	18630.057858	6.635246	333.775777	592.885359	15.180013	56.329076	4.500656	Not potable
2	8.099124	224.236259	19909.541732	9.275884	333.775777	418.606213	16.868637	66.420093	3.055934	Not potable
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771	Not potable
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	Not potable
...	...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821	Potable
3272	7.808856	193.553212	17329.802160	8.061362	333.775777	392.449580	19.903225	66.396293	2.798243	Potable
3273	9.419510	175.762646	33155.578218	7.350233	333.775777	432.044783	11.039070	69.845400	3.298875	Potable
3274	5.126763	230.603758	11983.869376	6.303357	333.775777	402.883113	11.168946	77.488213	4.708658	Potable
3275	7.874671	195.102299	17404.177061	7.509306	333.775777	327.459760	16.140368	78.698446	2.309149	Potable

3276 rows × 10 columns

12. Building a countpot to see the potability result distribution.

In [57]:
```
sns.countplot(x="Potability", data=train_data,
 palette="rainbow")
plt.title("Countplot based on Potable or Not potable");
```



13.Splitting our dataset for training

In [63]:
```
X = train_data.drop(['Potability'], axis=1)
X
```

Out[63]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity
0	7.080795	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135
1	3.716080	129.422921	18630.057858	6.635246	333.775777	592.885359	15.180013	56.329076	4.500656
2	8.099124	224.236259	19909.541732	9.275884	333.775777	418.606213	16.868637	66.420093	3.055934
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.341674	4.628771
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075
...	...	...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.687695	4.435821
3272	7.808856	193.553212	17329.802160	8.061362	333.775777	392.449580	19.903225	66.396293	2.798243
3273	9.419510	175.762646	33155.578218	7.350233	333.775777	432.044783	11.039070	69.845400	3.298875
3274	5.126763	230.603758	11983.869376	6.303357	333.775777	402.883113	11.168946	77.488213	4.708658
3275	7.874671	195.102299	17404.177061	7.509306	333.775777	327.459760	16.140368	78.698446	2.309149

3276 rows × 9 columns

In [64]:
```
y = train_data['Potability']
y
```

Out[64]:
```
0       Not potable
1       Not potable
2       Not potable
3       Not potable
4       Not potable
           ...
3271        Potable
3272        Potable
3273        Potable
3274        Potable
3275        Potable
Name: Potability, Length: 3276, dtype: category
Categories (2, object): ['Not potable', 'Potable']
```

In [65]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.30, random_state=1)
```

14.Printing all the shapes of our splitted dataset and start training our models.

In [66]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(2293, 9)
(983, 9)
(2293,)
(983,)
```

In [67]:
```
import sklearn.metrics as metrics
from sklearn.tree import DecisionTreeClassifier #for using Decision Tree Algoithm
model_dt = DecisionTreeClassifier(random_state=4)
model_dt.fit(X_train, y_train) #train the model with the training dataset
y_prediction_dt = model_dt.predict(X_test) #pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_dt = metrics.accuracy_score(y_prediction_dt, y_test).round(4)
print("---------------------------------")
print('The accuracy of the DT is: {}'.format(score_dt))
print("---------------------------------")
```

```
---------------------------------
The accuracy of the DT is: 0.5504
---------------------------------
```

In [68]:
```
from sklearn import svm #for Support Vector Machine (SVM) Algorithm
model_svm = svm.SVC() #select the algorithm
model_svm.fit(X_train, y_train) #train the model with the training dataset
y_prediction_svm = model_svm.predict(X_test) # pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_svm = metrics.accuracy_score(y_prediction_svm, y_test).round(4)
print("---------------------------------")
print('The accuracy of the SVM is: {}'.format(score_svm))
print("---------------------------------")
# save the accuracy score
score = set()
score.add(('SVM', score_svm))
```

```
---------------------------------
The accuracy of the SVM is: 0.5951
---------------------------------
```

In [69]:
```
# importing the necessary package to use the classification algorithm
from sklearn.neighbors import KNeighborsClassifier # for K nearest neighbours
#from sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm
model_knn = KNeighborsClassifier(n_neighbors=3) # 3 neighbours for putting the new data into a class
model_knn.fit(X_train, y_train) #train the model with the training dataset
y_prediction_knn = model_knn.predict(X_test) #pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_knn = metrics.accuracy_score(y_prediction_knn, y_test).round(4)
print("---------------------------------")
print('The accuracy of the KNN is: {}'.format(score_knn))
print("---------------------------------")
# save the accuracy score
score.add(('KNN', score_knn))
```

```
---------------------------------
The accuracy of the KNN is: 0.5453
---------------------------------
```

In [70]:
```
# importing the necessary package to use the classification algorithm
from sklearn.tree import DecisionTreeClassifier #for using Decision Tree Algoithm
model_dt = DecisionTreeClassifier(random_state=4)
model_dt.fit(X_train, y_train) #train the model with the training dataset
y_prediction_dt = model_dt.predict(X_test) #pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_dt = metrics.accuracy_score(y_prediction_dt, y_test).round(4)
print("---------------------------------")
print('The accuracy of the DT is: {}'.format(score_dt))
print("---------------------------------")
# save the accuracy score
score.add(('DT', score_dt))
```

```
---------------------------------
The accuracy of the DT is: 0.5504
---------------------------------
```

In [71]:
```
# importing the necessary package to use the classification algorithm
from sklearn.linear_model import LogisticRegression # for Logistic Regression algorithm
model_lr = LogisticRegression()
model_lr.fit(X_train, y_train) #train the model with the training dataset
y_prediction_lr = model_lr.predict(X_test) #pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_lr = metrics.accuracy_score(y_prediction_lr, y_test).round(4)
print("---------------------------------")
print('The accuracy of the LR is: {}'.format(score_lr))
print("---------------------------------")
# save the accuracy score
score.add(('LR', score_lr))
```

```
---------------------------------
The accuracy of the LR is: 0.5951
---------------------------------
```

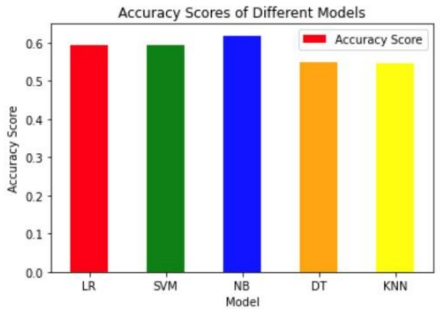
In [72]:
```
# importing the necessary package to use the classification algorithm
from sklearn.naive_bayes import GaussianNB
model_nb = GaussianNB()
model_nb.fit(X_train, y_train) #train the model with the training dataset
y_prediction_nb = model_nb.predict(X_test) #pass the testing data to the trained model
# checking the accuracy of the algorithm.
# by comparing predicted output by the model and the actual output
score_nb = metrics.accuracy_score(y_prediction_nb, y_test).round(4)
print("---------------------------------")
print('The accuracy of the NB is: {}'.format(score_nb))
print("---------------------------------")
# save the accuracy score
score.add(('NB', score_nb))
```

```
---------------------------------
The accuracy of the NB is: 0.6185
---------------------------------
```

In [73]:
```
print("The accuracy scores of different Models:")
print("----------------------------------------")
for s in score:
    print(s)
```

```
The accuracy scores of different Models:
----------------------------------------
('DT', 0.5504)
('KNN', 0.5453)
('LR', 0.5951)
('NB', 0.6185)
('SVM', 0.5951)
```

In [75]:
```
accuracy_scores = {
    'LR': 0.5951,
    'SVM': 0.5951,
    'NB': 0.6185,
    'DT': 0.5504,
    'KNN': 0.5453
}

colors = ['red', 'green', 'blue', 'orange','yellow']

df = pd.DataFrame(list(accuracy_scores.items()), columns=['Model', 'Accuracy Score'])

ax = df.plot.bar(x='Model', y='Accuracy Score', rot=0,color=colors)

ax.set_xlabel('Model')
ax.set_ylabel('Accuracy Score')
ax.set_title('Accuracy Scores of Different Models')
plt.show()
```



Conclusion:

We have done this project into jupyter notebook. We did analysis on water potability prediction. The dataset was collected from kaggle and it contains 3276 instances and 10 features. we developed 5 different classifier models which are Gaussian Naive Bayes, K Nearest Neighbors (KNN), Decision Tree, Logistic Regression and Support Vector Machine. Here we got the highest accuracy from Naive Bayes (NB) model which is almost 61% and the lowest accuracy we got from KNN model which is almost 54%. Logistic Regression (LR) and Support Vector Machine (SVM) models are given the same accuracy which are 59%.Decision Tree gives the accuracy of 55%.So based on the project we can say Naive Bayes performs best on the water potability prediction. Wecould gain more accuracy from other model as well if we had larger dataset.