In [1]:
```
#importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#
```

In [2]:
```
# reading dataset
df=pd.read_csv('D:\IGDTUW\WaterNet_Dataset\WaterNet_Dataset\DrinkingWater_Final_Dataset.csv')
print(df.shape)
```

```
(718, 14)
```

In [3]:
```
df.head()
```

Out[3]:

	id	pH	Sodium	Magnesium	Calcium	Chloride	Potassium	Carbonate	Sulphate	TDS	EC	TH	WQI	Potability
0	L1	7.26	77.51	32.55	81.4	63.55	1.95	419.68	68.16	640.0	1045.0	338.0	70.51	0
1	L2	7.54	36.11	31.10	42.4	19.53	1.56	273.28	57.12	400.0	645.0	234.0	48.31	1
2	L3	7.66	119.37	19.00	58.2	60.71	1.17	195.20	220.80	640.0	998.0	224.0	55.70	0
3	L4	5.98	117.07	29.65	145.4	37.28	2.73	580.72	190.08	950.0	1516.0	486.0	97.02	0
4	L5	7.02	45.08	23.35	49.6	23.43	1.17	248.88	74.88	410.0	658.0	220.0	46.45	1

In [4]:
```
df.describe()
```

Out[4]:

	pH	Sodium	Magnesium	Calcium	Chloride	Potassium	Carbonate	Sulphate	TDS	EC	TH	W
count	718.000000	718.000000	718.000000	718.000000	718.000000	718.000000	718.000000	718.000000	718.000000	718.000000	718.000000	718.0000
mean	7.677284	84.948579	56.268189	70.926671	119.842312	12.113496	117.960822	50.705153	499.612535	1012.396281	197.817298	64.1698
std	0.688892	147.306800	113.798579	141.330367	344.770005	17.270676	93.635222	78.488945	978.478140	1465.507731	154.203773	71.3791
min	2.410000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	2.100000	0.000000	8.3000
25%	7.270000	19.052500	14.000000	18.075000	17.750000	3.800000	54.612500	14.400000	0.000000	434.250000	116.247500	33.8350
50%	7.640000	38.875000	19.000000	28.850000	36.240000	6.935000	116.160000	35.040000	195.750000	543.400000	162.000000	40.1650
75%	8.110000	70.910000	36.825000	62.000000	76.600000	11.000000	144.980000	53.887500	696.500000	1020.000000	271.500000	63.8125
max	10.010000	1157.800000	990.000000	1158.000000	4700.000000	110.000000	580.720000	1159.000000	11619.000000	16598.000000	1400.000000	722.6900

In [5]:
```
df.dtypes
```

Out[5]:
```
id             object
pH            float64
Sodium        float64
Magnesium     float64
Calcium       float64
Chloride      float64
Potassium     float64
Carbonate     float64
Sulphate      float64
TDS           float64
EC            float64
TH            float64
WQI           float64
Potability      int64
dtype: object
```

In [7]:
```
# checking number of null values
df.isnull().sum()
```

Out[7]:
```
id            0
pH            0
Sodium        0
Magnesium     0
Calcium       0
Chloride      0
Potassium     0
Carbonate     0
Sulphate      0
TDS           0
EC            0
TH            0
WQI           0
Potability    0
dtype: int64
```

therefore, dataset has no null values. Also, the column id is redundant, so removing it won't affect the analysis.

In [6]:
```
df.drop(['id'],axis=1,inplace=True)
df.head()
```

Out[6]:

	pH	Sodium	Magnesium	Calcium	Chloride	Potassium	Carbonate	Sulphate	TDS	EC	TH	WQI	Potability
0	7.26	77.51	32.55	81.4	63.55	1.95	419.68	68.16	640.0	1045.0	338.0	70.51	0
1	7.54	36.11	31.10	42.4	19.53	1.56	273.28	57.12	400.0	645.0	234.0	48.31	1
2	7.66	119.37	19.00	58.2	60.71	1.17	195.20	220.80	640.0	998.0	224.0	55.70	0
3	5.98	117.07	29.65	145.4	37.28	2.73	580.72	190.08	950.0	1516.0	486.0	97.02	0
4	7.02	45.08	23.35	49.6	23.43	1.17	248.88	74.88	410.0	658.0	220.0	46.45	1

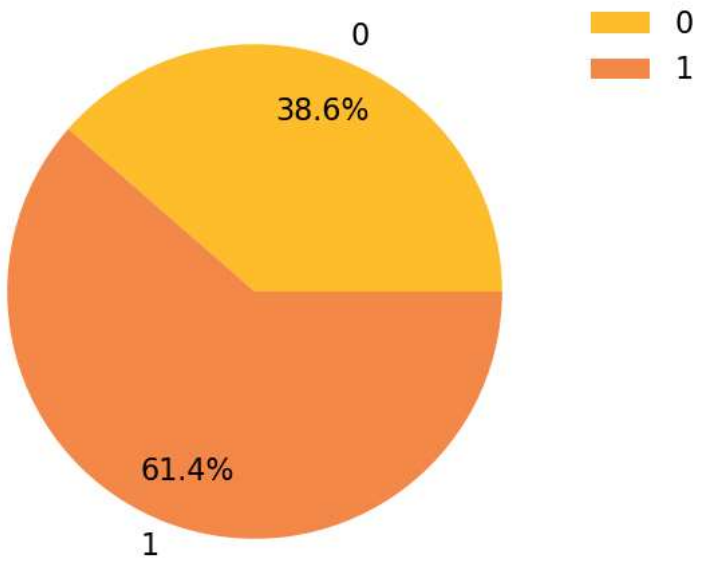
In [4]:
```
#Looking at number of output values for dependent variable Potability
df['Potability'].value_counts()
```

Out[4]:
```
1    441
0    277
Name: Potability, dtype: int64
```

In [11]:
```
counting=list(df.groupby('Potability')['pH'].count())
print(counting)
```

```
[277, 441]
```

In [13]:
```
pal_ = list(sns.color_palette(palette='plasma_r', n_colors=6).as_hex())
plt.figure(figsize=(6, 6))
plt.rcParams.update({'font.size': 16})
plt.pie(counting,
        labels=[0,1],autopct='%1.1f%%',
        pctdistance=0.78,colors=pal_)
plt.legend(bbox_to_anchor=(1, 1), loc=2, frameon=False)
plt.savefig("piechart.pdf", format="pdf", bbox_inches="tight")
plt.show()
```

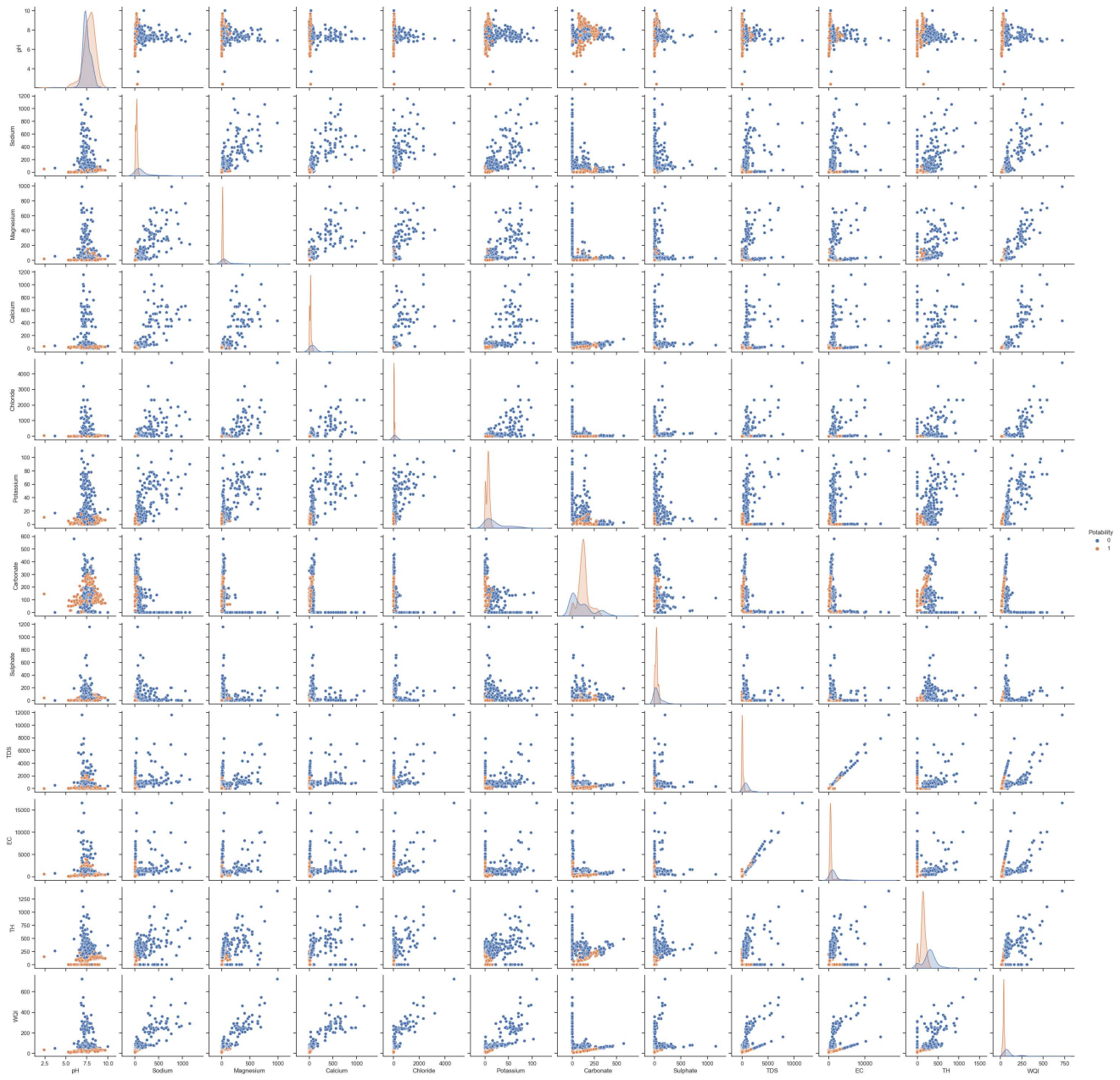


In [5]:
```
print("Proportion of true value is {}".format(441/(441+277)))
print("Proportion of false value is {}".format(277/(441+277)))
```

```
Proportion of true value is 0.6142061281337048
Proportion of false value is 0.38579387186629527
```

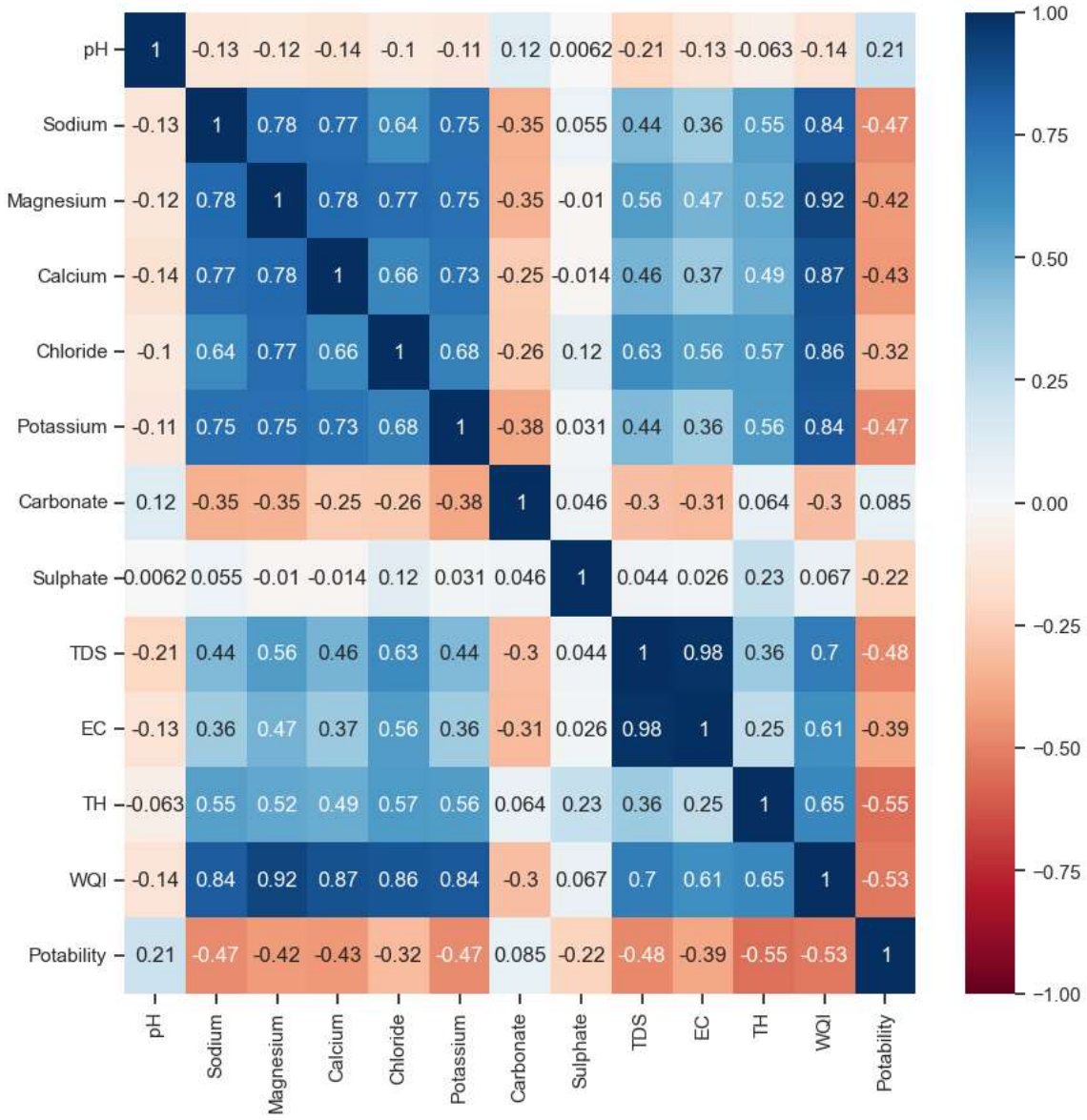
In [60]:
```
sns.set(style='ticks',color_codes=True)
sns.pairplot(df,hue="Potability")
```

Out[60]: <seaborn.axisgrid.PairGrid at 0x12f4c858d30>



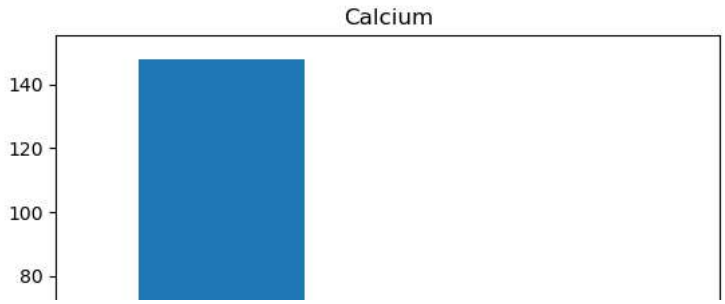
In [70]:
```
plt.figure(figsize=(10,10))
sns.heatmap(df.corr(),cmap='RdBu',annot=True,vmin=-1,vmax=1)
```

Out[70]: <AxesSubplot:>



The dataset seems to be fairly balanced as ratio of true values to false values is roughly 3:2.

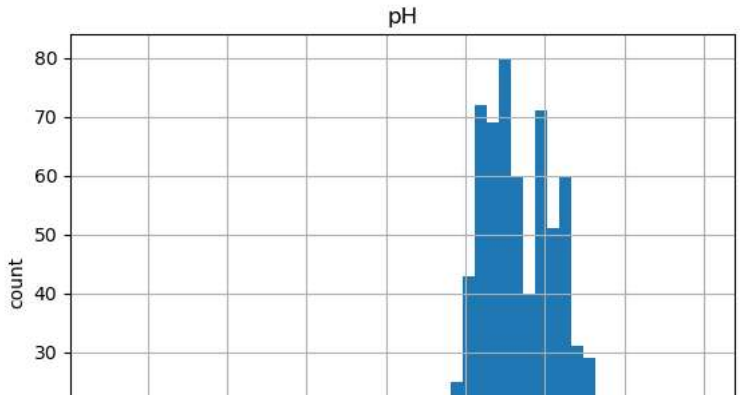
In [11]:
```
for feature in df:
    data=df.copy()
    data.groupby(['Potability'])[feature].mean().plot.bar()
    plt.title(feature)
    plt.show()
```



We can see that relatively low mean levels of Sodium, magnesium, calcium, chloride, Potassium and TDS makes the water Potable.

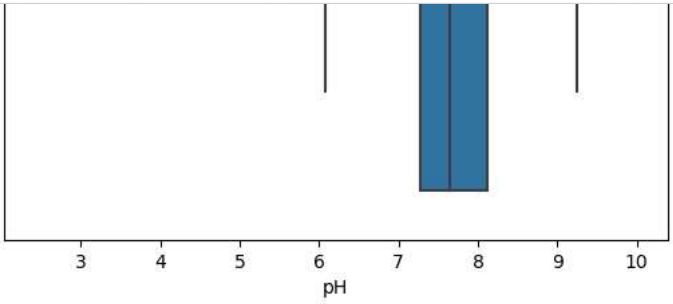
In [12]:
```
for feature in df:
    data=df.copy()
    print("Number of distinct values for {} are {}".format(feature,len(data[feature].unique())))
    data[feature].hist(bins=50)
    plt.xlabel(feature)
    plt.ylabel('count')
    plt.title(feature)
    plt.show()
```

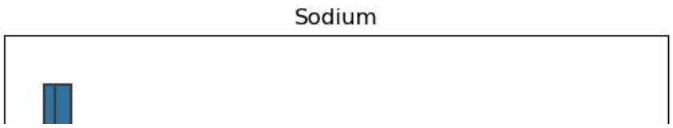
```
Number of distinct values for pH are 239
```



In [15]:
```
feature_dataset=df.drop(['Potability'],axis=1)
```

In [17]:
```
for feature in data:
    data=df.copy()
    sns.boxplot(x=data[feature])
    plt.xlabel(feature)
    plt.ylabel('')
    plt.title(feature)
    plt.show()
```





In [16]:
```
#Tukey's method
def tukeys_method(df, variable):
    #Takes two parameters: dataframe & variable of interest as string
    q1 = df[variable].quantile(0.25)
    q3 = df[variable].quantile(0.75)
    iqr = q3-q1
    inner_fence = 1.5*iqr
    outer_fence = 3*iqr

    #inner fence lower and upper end
    inner_fence_le = q1-inner_fence
    inner_fence_ue = q3+inner_fence

    #outer fence lower and upper end
    outer_fence_le = q1-outer_fence
    outer_fence_ue = q3+outer_fence

    outliers_prob = []
    outliers_poss = []
    for index, x in enumerate(df[variable]):
        if x <= outer_fence_le or x >= outer_fence_ue:
            outliers_prob.append(index)
    for index, x in enumerate(df[variable]):
        if x <= inner_fence_le or x >= inner_fence_ue:
            outliers_poss.append(index)
    return outliers_prob, outliers_poss
for x in feature_dataset:
    probable_outliers_tm, possible_outliers_tm = tukeys_method(feature_dataset,x)
    print("Outliers for {} are {}".format(x,probable_outliers_tm))
    print("Outliers for {} are {}".format(x,possible_outliers_tm))
```

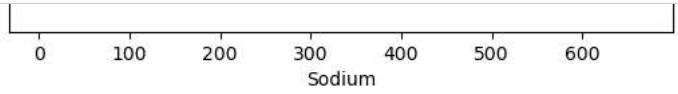
```
Outliers for pH are [172, 420]
Outliers for pH are [3, 172, 178, 285, 286, 369, 420, 702, 703, 706, 709, 711, 712, 713, 714, 715, 716, 717]
Outliers for Sodium are [90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 11
2, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 128, 129, 130, 131, 132, 133, 134, 136, 137, 139, 140,
141, 142, 143, 144, 145, 147, 610, 625, 644, 648, 653, 658, 660, 671]
Outliers for Sodium are [90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 11
2, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137,
138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 149, 150, 152, 155, 161, 162, 163, 164, 167, 174, 178, 594, 598, 604, 606, 60
8, 610, 625, 626, 629, 644, 646, 647, 648, 653, 654, 658, 660, 669, 671, 676, 679, 684]
Outliers for Magnesium are [90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111,
112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 13
7, 138, 139, 140, 141, 142, 143, 144, 145, 150, 151, 152, 157, 159, 160, 162, 163, 164, 165, 166, 169, 170, 212, 221, 222, 233,
246, 247, 248]
Outliers for Magnesium are [90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111,
112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 13
7, 138, 139, 140, 141, 142, 143, 144, 145, 150, 151, 152, 153, 154, 155, 156, 157, 159, 160, 162, 163, 164, 165, 166, 167, 168,
169, 170, 171, 208, 212, 221, 222, 223, 233, 235, 238, 246, 247, 248, 252, 256, 264, 268, 274, 651]
Outliers for Calcium are [90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 1
12, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 136, 137, 13
8, 139, 140, 142, 145]
Outliers for Calcium are [3, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 11
1, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 136, 137,
138, 139, 140, 141, 142, 144, 145, 152, 174]
Outliers for Chloride are [90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 107, 108, 109, 110, 111, 112, 113,
116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 129, 130, 132, 133, 134, 135, 136, 137, 139, 140, 612, 613, 615, 61
8, 621, 662, 663, 665, 668]
Outliers for Chloride are [90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 107, 108, 109, 110, 111, 112, 113,
116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 139, 140, 141, 59
5, 596, 597, 598, 603, 608, 612, 613, 614, 615, 616, 618, 621, 622, 623, 630, 631, 642, 645, 646, 647, 648, 651, 653, 658, 662,
663, 664, 665, 666, 668, 671, 672, 673, 674, 681, 692]
Outliers for Potassium are [90, 91, 92, 93, 94, 95, 96, 98, 99, 100, 101, 102, 103, 104, 106, 107, 108, 109, 110, 111, 112, 11
3, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 133, 134, 135, 136, 137, 138, 139,
140, 141, 142, 143, 144, 614, 626, 641, 646, 652, 677, 678, 679, 680, 681, 684, 685, 687, 688, 689, 690]
Outliers for Potassium are [90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111,
112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 13
7, 138, 139, 140, 141, 142, 143, 144, 148, 150, 154, 155, 157, 162, 163, 165, 166, 170, 175, 178, 596, 602, 605, 606, 610, 614,
620, 626, 641, 644, 645, 646, 651, 652, 655, 656, 664, 669, 674, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 68
9, 690, 691]
Outliers for Carbonate are [0, 3, 11, 87, 581, 591]
Outliers for Carbonate are [0, 3, 5, 7, 8, 9, 10, 11, 13, 14, 15, 24, 30, 31, 32, 34, 35, 38, 40, 43, 47, 48, 49, 50, 52, 55, 5
7, 59, 60, 64, 66, 67, 69, 71, 72, 73, 74, 75, 76, 77, 80, 83, 84, 87, 88, 89, 231, 577, 579, 581, 582, 583, 584, 586, 588, 59
0, 591, 592]
Outliers for Sulphate are [2, 3, 6, 20, 22, 28, 94, 121, 593, 597, 598, 599, 603, 610, 615, 616, 617, 618, 620, 625, 643, 646,
647, 648, 649, 653, 660, 664, 665, 666, 667, 668, 670, 672, 675]
Outliers for Sulphate are [2, 3, 6, 20, 22, 27, 28, 94, 101, 107, 110, 120, 121, 229, 397, 593, 595, 597, 598, 599, 600, 601, 6
03, 610, 614, 615, 616, 617, 618, 619, 620, 622, 623, 624, 625, 631, 643, 645, 646, 647, 648, 649, 650, 651, 652, 653, 660, 66
4, 665, 666, 667, 668, 669, 670, 672, 673, 674, 675]
Outliers for TDS are [93, 94, 99, 101, 107, 110, 113, 120, 121, 528, 536, 539, 540, 544, 548, 561, 565, 574, 575]
Outliers for TDS are [92, 93, 94, 96, 99, 101, 102, 107, 109, 110, 113, 114, 116, 117, 118, 120, 121, 527, 528, 536, 538, 539,
540, 544, 548, 549, 550, 554, 556, 559, 561, 562, 565, 567, 570, 571, 572, 574, 575, 576, 613, 663]
Outliers for EC are [92, 93, 94, 96, 99, 101, 107, 109, 110, 113, 114, 116, 117, 118, 120, 121, 527, 528, 529, 535, 536, 538, 5
39, 540, 544, 548, 549, 550, 554, 556, 559, 561, 562, 565, 567, 570, 571, 572, 573, 574, 575, 576, 613, 663]
Outliers for EC are [91, 92, 93, 94, 95, 96, 99, 100, 101, 102, 106, 107, 109, 110, 113, 114, 115, 116, 117, 118, 120, 121, 52
7, 528, 529, 530, 531, 534, 535, 536, 538, 539, 540, 543, 544, 545, 548, 549, 550, 552, 553, 554, 556, 557, 558, 559, 560, 561,
562, 563, 564, 565, 566, 567, 568, 570, 571, 572, 573, 574, 575, 576, 613, 663]
Outliers for TH are [94, 101, 113, 116, 117, 118, 120, 141]
Outliers for TH are [92, 94, 99, 101, 103, 104, 107, 108, 113, 114, 115, 116, 117, 118, 120, 139, 140, 141, 153, 155, 158, 601,
651]
Outliers for WQI are [90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 13
8, 139, 140, 141, 142, 544]
Outliers for WQI are [90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112,
113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 13
8, 139, 140, 141, 142, 143, 144, 145, 528, 544, 685, 687]
```

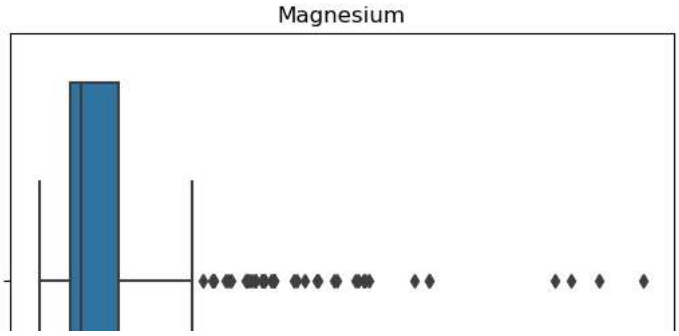
In [17]:
```
for x in feature_dataset:
    q1 = feature_dataset[x].quantile(0.25)
    q3 = feature_dataset[x].quantile(0.75)
    iqr = q3-q1
    inner_fence = 1.5*iqr
    outer_fence = 3*iqr

    #inner fence lower and upper end
    inner_fence_le = q1-inner_fence
    inner_fence_ue = q3+inner_fence

    #outer fence lower and upper end
    outer_fence_le = q1-outer_fence
    outer_fence_ue = q3+outer_fence
    for i in (probable_outliers_tm or possible_outliers_tm)  :
            if feature_dataset[x].iloc[i]>=inner_fence_ue:
                feature_dataset[x].iloc[i]=inner_fence_ue
            elif feature_dataset[x].iloc[i]<=inner_fence_le:
                 feature_dataset[x].iloc[i] =inner_fence_le
```

In [8]:
```
for feature in feature_dataset:
    data=feature_dataset.copy()
    sns.boxplot(x=data[feature])
    plt.xlabel(feature)
    plt.ylabel('')
    plt.title(feature)
    plt.show()
```





In [18]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import VotingClassifier
from sklearn import model_selection
from sklearn.metrics import confusion_matrix
from sklearn. preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

In [19]:
```
import sklearn.metrics as metrics
def reg_results(y,yhat):
    mae = metrics.mean_absolute_error(y, yhat)
    mse = metrics.mean_squared_error(y, yhat)
    rmse = np.sqrt(mse) # or mse**(0.5)
    r2 = metrics.r2_score(y,yhat)

    print("MAE:",mae)
    print("MSE:", mse)
    print("RMSE:", rmse)
    print("R-Squared:", r2)
    return True
```

In [20]:
```
from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
def res(y,yhat):
    print(confusion_matrix(y,yhat))
    print(accuracy_score(y,yhat))
    print(classification_report(y,yhat))
```

In [21]:
```
Y=df['Potability']
X=feature_dataset
```

In [22]:
```
X_train,X_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=0)
```

In [23]:
```
print(X.shape)
print(Y.shape)
print(X_train.shape)
print(X_test.shape)

print(y_train.shape)
print(y_test.shape)
```

```
(718, 12)
(718,)
(538, 12)
(180, 12)
(538,)
(180,)
```

Random forest

In [77]:
```
#Fitting Decision Tree classifier to the training set
from sklearn.ensemble import RandomForestClassifier
classifier= RandomForestClassifier(n_estimators= 10, criterion="entropy")
classifier.fit(X_train, y_train)
```

Out[77]:
```
RandomForestClassifier(criterion='entropy', n_estimators=10)
```

In [78]:
```
y_pred= classifier.predict(X_test)
reg_results(y_test,y_pred)
res(y_test,y_pred)
ytest_pred4 = classifier.predict_proba(X_test)
```

```
MAE: 0.0
MSE: 0.0
RMSE: 0.0
R-Squared: 1.0
[[ 55   0]
 [  0 125]]
1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        55
           1       1.00      1.00      1.00       125

    accuracy                           1.00       180
   macro avg       1.00      1.00      1.00       180
weighted avg       1.00      1.00      1.00       180
```

Ada boost

In [26]:
```
from sklearn.ensemble import AdaBoostClassifier
ada_classifier=AdaBoostClassifier()
ada_classifier.fit(X_train, y_train)
ytrain_pred = ada_classifier.predict_proba(X_train)
print('Adaboost train roc-auc: {}'.format(roc_auc_score(y_train, ytrain_pred[:,1])))
ytest_pred = ada_classifier.predict_proba(X_test)
print('Adaboost test roc-auc: {}'.format(roc_auc_score(y_test, ytest_pred[:,1])))
y_pred=ada_classifier.predict(X_test)
reg_results(y_test,y_pred)
res(y_test,y_pred)
```

```
Adaboost train roc-auc: 1.0
Adaboost test roc-auc: 1.0
MAE: 0.0
MSE: 0.0
RMSE: 0.0
R-Squared: 1.0
[[ 55   0]
 [  0 125]]
1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        55
           1       1.00      1.00      1.00       125

    accuracy                           1.00       180
   macro avg       1.00      1.00      1.00       180
weighted avg       1.00      1.00      1.00       180
```

KNN

In [71]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_classifier=KNeighborsClassifier()
knn_classifier.fit(X_train, y_train)
ytrain_pred = knn_classifier.predict_proba(X_train)
print('KNN train roc-auc: {}'.format(roc_auc_score(y_train, ytrain_pred[:,1])))
ytest_pred1 = knn_classifier.predict_proba(X_test)
print('KNN test roc-auc: {}'.format(roc_auc_score(y_test, ytest_pred1[:,1])))
y_pred=knn_classifier.predict(X_test)
reg_results(y_test,y_pred)
res(y_test,y_pred)
```

```
KNN train roc-auc: 0.9983678298551717
KNN test roc-auc: 0.9948363636363637
MAE: 0.044444444444444446
MSE: 0.044444444444444446
RMSE: 0.21081851067789195
R-Squared: 0.7905454545454546
[[ 53   2]
 [  6 119]]
0.9555555555555556
              precision    recall  f1-score   support

           0       0.90      0.96      0.93        55
           1       0.98      0.95      0.97       125

    accuracy                           0.96       180
   macro avg       0.94      0.96      0.95       180
weighted avg       0.96      0.96      0.96       180
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction func
tions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, th
is behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will b
e eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
```

In [57]:
```
#Visulaizing the trianing set result
from matplotlib.colors import ListedColormap
x_set, y_set = X_train, y_train
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
plt.contourf(x1, x2, knn_classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN Algorithm (Training set)')
plt.xlabel('Age')
plt.ylabel('Potability')
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3628             try:
-> 3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:

C:\ProgramData\Anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

C:\ProgramData\Anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

TypeError: '(slice(None, None, None), 0)' is an invalid key

During handling of the above exception, another exception occurred:

InvalidIndexError                         Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_148312\3882566028.py in <module>
      2 from matplotlib.colors import ListedColormap
      3 x_set, y_set = X_train, y_train
----> 4 x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
      5 np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
      6 plt.contourf(x1, x2, knn_classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   3503             if self.columns.nlevels > 1:
   3504                 return self._getitem_multilevel(key)
-> 3505             indexer = self.columns.get_loc(key)
   3506             if is_integer(indexer):
   3507                 indexer = [indexer]

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3634                 #  InvalidIndexError. Otherwise we fall through and re-raise
   3635                 #  the TypeError.
-> 3636                 self._check_indexing_error(key)
   3637                 raise
   3638

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in _check_indexing_error(self, key)
   5649             # if key is not a scalar, directly raise an error (the code below
   5650             # would convert to numpy arrays and raise later any way) - GH29926
-> 5651             raise InvalidIndexError(key)
   5652
   5653     @cache_readonly

InvalidIndexError: (slice(None, None, None), 0)
```

In [59]:
```
#Visualizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = X_test, y_test
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
plt.contourf(x1, x2, knn_classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('K-NN algorithm(Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3628             try:
-> 3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:

C:\ProgramData\Anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

C:\ProgramData\Anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

TypeError: '(slice(None, None, None), 0)' is an invalid key

During handling of the above exception, another exception occurred:

InvalidIndexError                         Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_148312\2205528317.py in <module>
      2 from matplotlib.colors import ListedColormap
      3 x_set, y_set = X_test, y_test
----> 4 x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
      5 np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
      6 plt.contourf(x1, x2, knn_classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   3503             if self.columns.nlevels > 1:
   3504                 return self._getitem_multilevel(key)
-> 3505             indexer = self.columns.get_loc(key)
   3506             if is_integer(indexer):
   3507                 indexer = [indexer]

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3634                 #  InvalidIndexError. Otherwise we fall through and re-raise
   3635                 #  the TypeError.
-> 3636                 self._check_indexing_error(key)
   3637                 raise
   3638

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in _check_indexing_error(self, key)
   5649             # if key is not a scalar, directly raise an error (the code below
   5650             # would convert to numpy arrays and raise later any way) - GH29926
-> 5651             raise InvalidIndexError(key)
   5652
   5653     @cache_readonly

InvalidIndexError: (slice(None, None, None), 0)
```

In [ ]:
```

```

Gradient boosting

In [73]:
```
from sklearn.ensemble import GradientBoostingClassifier
gbc=GradientBoostingClassifier(n_estimators=500,learning_rate=0.05,random_state=100,max_features=5 )

# Fit train data to GBC

gbc.fit(X_train,y_train)

print("GBC accuracy is %2.2f" % accuracy_score(
     y_test, gbc.predict(X_test)))

ytest_pred2 = gbc.predict_proba(X_test)
pred=gbc.predict(X_test)
reg_results(y_test,pred)
res(y_test,pred)
# print(classification_report(y_test, pred))
```

```
GBC accuracy is 1.00
MAE: 0.0
MSE: 0.0
RMSE: 0.0
R-Squared: 1.0
[[ 55   0]
 [  0 125]]
1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        55
           1       1.00      1.00      1.00       125

    accuracy                           1.00       180
   macro avg       1.00      1.00      1.00       180
weighted avg       1.00      1.00      1.00       180
```

Decision Tree

In [29]:
```
#Fitting Decision Tree classifier to the training set
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(X_train, y_train)
#Predicting the test set result
y_pred= classifier.predict(X_test)
reg_results(y_test,y_pred)
res(y_test,y_pred)

print("DT model accuracy(in %):", metrics.accuracy_score(y_test, y_pred)*100)
```

```
MAE: 0.0
MSE: 0.0
RMSE: 0.0
R-Squared: 1.0
[[ 55   0]
 [  0 125]]
1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        55
           1       1.00      1.00      1.00       125

    accuracy                           1.00       180
   macro avg       1.00      1.00      1.00       180
weighted avg       1.00      1.00      1.00       180

DT model accuracy(in %): 100.0
```

Gaussian Naive Bayes classifier

In [31]:
```
# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# making predictions on the testing set
y_pred = gnb.predict(X_test)
reg_results(y_test,y_pred)
res(y_test,y_pred)
```

```
MAE: 0.07777777777777778
MSE: 0.07777777777777778
RMSE: 0.27888667551135854
R-Squared: 0.6334545454545455
[[ 49   6]
 [  8 117]]
0.9222222222222223
              precision    recall  f1-score   support

           0       0.86      0.89      0.88        55
           1       0.95      0.94      0.94       125

    accuracy                           0.92       180
   macro avg       0.91      0.91      0.91       180
weighted avg       0.92      0.92      0.92       180
```

In [32]:
```
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(y_test, y_pred)*100)
```

```
Gaussian Naive Bayes model accuracy(in %): 92.22222222222223
```

In [33]:
```
#dropping a correlated feature
dnb=df.drop(['EC'],axis=1)
Xnb_train,Xnb_test,ynb_train,ynb_test=train_test_split(dnb.drop(['Potability'],axis=1),dnb['Potability'],test_size=0.25,random_st
```

In [34]:
```
# training the model on training set
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(Xnb_train, ynb_train)

# making predictions on the testing set
ynb_pred = gnb.predict(Xnb_test)
reg_results(ynb_test,ynb_pred)
res(ynb_test,ynb_pred)
```

```
MAE: 0.07222222222222222
MSE: 0.07222222222222222
RMSE: 0.26874192494328497
R-Squared: 0.6596363636363636
[[ 43  12]
 [  1 124]]
0.9277777777777778
              precision    recall  f1-score   support

           0       0.98      0.78      0.87        55
           1       0.91      0.99      0.95       125

    accuracy                           0.93       180
   macro avg       0.94      0.89      0.91       180
weighted avg       0.93      0.93      0.93       180
```

In [35]:
```
print("Gaussian Naive Bayes model accuracy(in %):", metrics.accuracy_score(ynb_test, ynb_pred)*100)
```

```
Gaussian Naive Bayes model accuracy(in %): 92.77777777777779
```

Logistic regression without scaling

In [74]:
```
from sklearn.linear_model import LogisticRegression
log_classifier=LogisticRegression()
log_classifier.fit(X_train, y_train)
ytrain_pred = log_classifier.predict_proba(X_train)
y_pred=log_classifier.predict(X_test)
reg_results(y_test,y_pred)
res(y_test,y_pred)
print("Accuracy score is:",metrics.accuracy_score(y_test,y_pred))
print('Logistic train roc-auc: {}'.format(roc_auc_score(y_train, ytrain_pred[:,1])))
ytest_pred3 = log_classifier.predict_proba(X_test)
print('Logistic test roc-auc: {}'.format(roc_auc_score(y_test, ytest_pred[:,1])))
```

```
MAE: 0.07777777777777778
MSE: 0.07777777777777778
RMSE: 0.27888667551135854
R-Squared: 0.6334545454545455
[[ 49   6]
 [  8 117]]
0.9222222222222223
              precision    recall  f1-score   support

           0       0.86      0.89      0.88        55
           1       0.95      0.94      0.94       125

    accuracy                           0.92       180
   macro avg       0.91      0.91      0.91       180
weighted avg       0.92      0.92      0.92       180

Accuracy score is: 0.9222222222222223
Logistic train roc-auc: 0.9914756528680579
Logistic test roc-auc: 0.9825454545454545

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge
(status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/modules/line
ar_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(
```

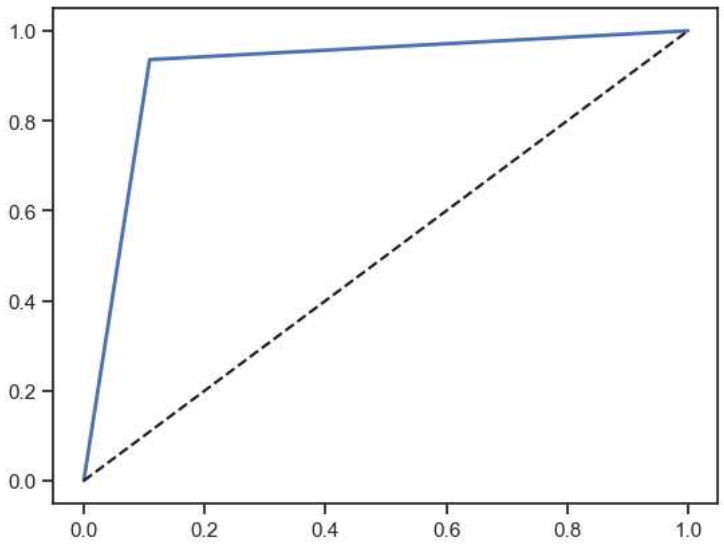
multiple linear regression witout scaling

In [76]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
reg_results(y_test,y_pred)
```

```
MAE: 0.18298372478276398
MSE: 0.0585217331258818
RMSE: 0.2419126559853407
R-Squared: 0.7242030322503897
```

Out[76]: True

In [68]:
```
# Roc curve
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
plt.plot(fpr, tpr, linewidth=2)
plt.plot([0, 1], [0, 1], 'k--')
plt.show()
```



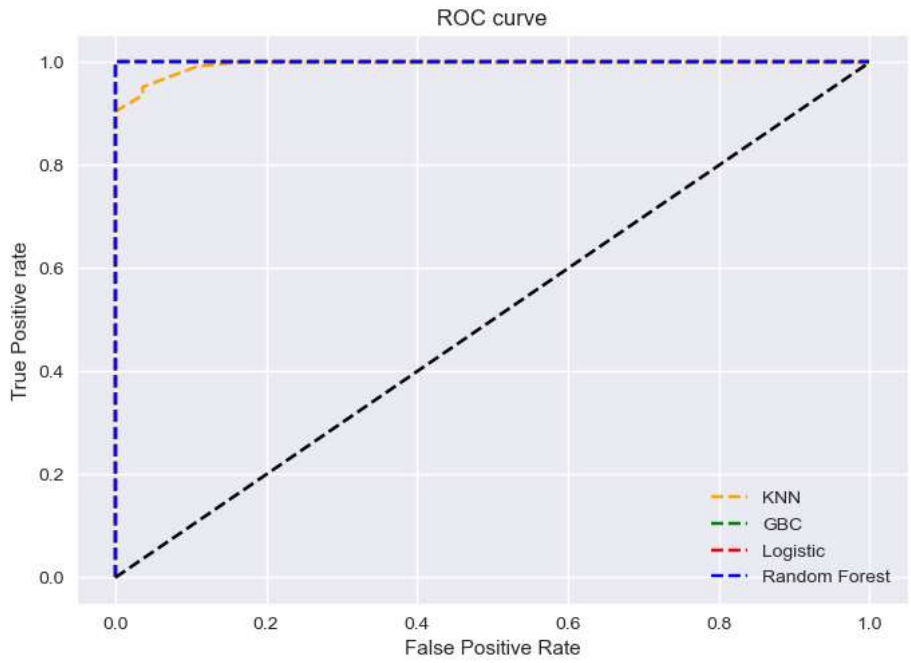
In [79]:
```
from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(y_test, ytest_pred1[:,1], pos_label=1)
fpr2, tpr2, thresh2 = roc_curve(y_test, ytest_pred2[:,1], pos_label=1)
fpr3, tpr3, thresh3 = roc_curve(y_test, ytest_pred3[:,1], pos_label=1)
fpr4, tpr4, thresh4 = roc_curve(y_test, ytest_pred4[:,1], pos_label=1)
# matplotlib
import matplotlib.pyplot as plt
# roc curve for tpr = fpr
random_probs = [0 for i in range(len(y_test))]
p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)

plt.style.use('seaborn')

# plot roc curves
plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='KNN')
plt.plot(fpr2, tpr2, linestyle='--',color='green', label='GBC')
plt.plot(fpr2, tpr2, linestyle='--',color='red', label='Logistic')
plt.plot(fpr2, tpr2, linestyle='--',color='blue', label='Random Forest')
plt.plot(p_fpr, p_tpr, linestyle='--', color='black')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```



Robust scaling

In [36]:
```
from sklearn.preprocessing import RobustScaler
rs=RobustScaler()
rs_data=rs.fit_transform(feature_dataset)
rs_df=pd.DataFrame(rs_data)
rs_df.columns=feature_dataset.columns
```

In [37]:
```
rs_df.head()
```

Out[37]:

	pH	Sodium	Magnesium	Calcium	Chloride	Potassium	Carbonate	Sulphate	TDS	EC	TH	WQI
0	-0.452381	0.745022	0.593647	1.196357	0.464061	-0.692361	3.358730	0.838746	0.637832	0.856338	1.133637	1.012259
1	-0.119048	-0.053319	0.530120	0.308480	-0.283942	-0.746528	1.738678	0.559164	0.293252	0.173453	0.463761	0.271704
2	0.023810	1.552234	0.000000	0.668184	0.415803	-0.800694	0.874651	4.704274	0.637832	0.776099	0.399349	0.518222
3	-1.976190	1.507882	0.466594	2.653386	0.017672	-0.584028	5.140786	3.926306	1.082915	1.660435	2.086923	1.896589
4	-0.738095	0.119655	0.190581	0.472396	-0.217672	-0.800694	1.468670	1.008927	0.307609	0.195647	0.373585	0.209657

In [38]:
```
Xs_train,Xs_test,ys_train,ys_test=train_test_split(rs_df,df['Potability'],test_size=0.25,random_state=0)
```

Multiple Linear Regression

In [44]:
```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(Xs_train, ys_train)
```

Out[44]:
```
LinearRegression()
```

In [45]:
```
y_pred = regressor.predict(Xs_test)
reg_results(ys_test,y_pred)
```

```
MAE: 0.18298372478276406
MSE: 0.05852173312588187
RMSE: 0.24191265598534084
R-Squared: 0.7242030322503894
```

Out[45]:
```
True
```

polynomial regression

In [54]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
poly_reg = PolynomialFeatures(degree = 4)
X_poly = poly_reg.fit_transform(Xs_train)
poly_reg.fit(X_poly,ys_train)
# lin_reg = LinearRegression()
# lin_reg.fit(X_poly, ys_train)
```

Out[54]:
```
PolynomialFeatures(degree=4)
```

In [55]:
```
y_pred=poly_reg.predict(Xs_test)
# y_pred = lin_reg.predict(Xs_test)
reg_results(ys_test,y_pred)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_148312\3343446894.py in <module>
----> 1 y_pred=poly_reg.predict(Xs_test)
      2 # y_pred = lin_reg.predict(Xs_test)
      3 reg_results(ys_test,y_pred)

AttributeError: 'PolynomialFeatures' object has no attribute 'predict'
```

logistic

In [62]:
```
from sklearn.linear_model import LogisticRegression
log_classifier=LogisticRegression()
log_classifier.fit(Xs_train, ys_train)
ytrain_pred = log_classifier.predict_proba(Xs_train)
y_pred=log_classifier.predict(Xs_test)
reg_results(ys_test,y_pred)
res(ys_test,y_pred)
print("Accuracy score is:",metrics.accuracy_score(ys_test,y_pred))
print('Logistic train roc-auc: {}'.format(roc_auc_score(ys_train, ytrain_pred[:,1])))
ytest_pred = log_classifier.predict_proba(Xs_test)
print('Logistic test roc-auc: {}'.format(roc_auc_score(ys_test, ytest_pred[:,1])))
```

```
MAE: 0.005555555555555556
MSE: 0.005555555555555556
RMSE: 0.07453559924999299
R-Squared: 0.9738181818181818
[[ 55   0]
 [  1 124]]
0.9944444444444445
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        55
           1       1.00      0.99      1.00       125

    accuracy                           0.99       180
   macro avg       0.99      1.00      0.99       180
weighted avg       0.99      0.99      0.99       180

Accuracy score is: 0.9944444444444445
Logistic train roc-auc: 1.0
Logistic test roc-auc: 1.0
```

ridge regression

In [52]:
```
from sklearn.linear_model import Ridge

## training the model

ridgeReg = Ridge(alpha=0.05, normalize=True)

ridgeReg.fit(Xs_train,ys_train)

pred = ridgeReg.predict(Xs_test)
reg_results(ys_test,pred)
print(ridgeReg.score(Xs_test,ys_test))
```

```
MAE: 0.19509225264043428
MSE: 0.062247788848376
RMSE: 0.2494950677836658
R-Squared: 0.7066431478272899
0.7066431478272899

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_base.py:141: FutureWarning: 'normalize' was deprecated in vers
ion 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), Ridge())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * n_samples.
  warnings.warn(
```

Lasso

In [63]:
```
from sklearn.linear_model import Lasso

lassoReg = Lasso(alpha=0.3, normalize=True)

lassoReg.fit(Xs_train,ys_train)

pred = lassoReg.predict(Xs_test)
reg_results(ys_test,pred)
print(lassoReg.score(Xs_test,ys_test))
```

```
MAE: 0.4660264353572903
MSE: 0.2236583088803207
RMSE: 0.47292526775413546
R-Squared: -0.05404061203234778
-0.05404061203234778

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_base.py:141: FutureWarning: 'normalize' was deprecated in vers
ion 1.0 and will be removed in 1.2.
If you wish to scale the data, use Pipeline with a StandardScaler in a preprocessing stage. To reproduce the previous behavior:

from sklearn.pipeline import make_pipeline

model = make_pipeline(StandardScaler(with_mean=False), Lasso())

If you wish to pass a sample_weight parameter, you need to pass it as a fit parameter to each step of the pipeline as follows:

kwargs = {s[0] + '__sample_weight': sample_weight for s in model.steps}
model.fit(X, y, **kwargs)

Set parameter alpha to: original_alpha * np.sqrt(n_samples).
  warnings.warn(
```

In [ ]:
```
```