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

In [2]:
```python
df=pd.read_csv("C:\\Users\\Asus\\Desktop\\water_potability.csv")
df
```

Out[2]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279
...	...	...	...	...	...	...	...
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419
3272	7.808856	193.553212	17329.802160	8.061362	NaN	392.449580	19.903225
3273	9.419510	175.762646	33155.578218	7.350233	NaN	432.044783	11.039070
3274	5.126763	230.603758	11983.869376	6.303357	NaN	402.883113	11.168946
3275	7.874671	195.102299	17404.177061	7.509306	NaN	327.459760	16.140368

3276 rows × 10 columns

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

Out[3]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Tr
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	

In [4]:
```python
df.info
```

Out[4]:
```
<bound method DataFrame.info of             ph    Hardness        Solids
Chloramines     Sulfate  \
0          NaN  204.890455  20791.318981     7.300212  368.516441
1     3.716080  129.422921  18630.057858     6.635246         NaN
2     8.099124  224.236259  19909.541732     9.275884         NaN
3     8.316766  214.373394  22018.417441     8.059332  356.886136
4     9.092223  181.101509  17978.986339     6.546600  310.135738
...        ...         ...           ...          ...         ...
3271  4.668102  193.681735  47580.991603     7.166639  359.948574
3272  7.808856  193.553212  17329.802160     8.061362         NaN
3273  9.419510  175.762646  33155.578218     7.350233         NaN
3274  5.126763  230.603758  11983.869376     6.303357         NaN
3275  7.874671  195.102299  17404.177061     7.509306         NaN

      Conductivity  Organic_carbon  Trihalomethanes  Turbidity  Potabili
ty
0       564.308654       10.379783        86.990970   2.963135
0
1       592.885359       15.180013        56.329076   4.500656
0
2       418.606213       16.868637        66.420093   3.055934
0
3       363.266516       18.436524       100.341674   4.628771
0
4       398.410813       11.558279        31.997993   4.075075
0
...            ...             ...              ...        ...
...
3271    526.424171       13.894419        66.687695   4.435821
1
3272    392.449580       19.903225              NaN   2.798243
1
3273    432.044783       11.039070        69.845400   3.298875
1
3274    402.883113       11.168946        77.488213   4.708658
1
3275    327.459760       16.140368        78.698446   2.309149
1

[3276 rows x 10 columns]>
```

In [5]:
```python
df.describe
```

Out[5]:
```
<bound method NDFrame.describe of             ph    Hardness        Soli
ds  Chloramines     Sulfate  \
0          NaN  204.890455  20791.318981     7.300212  368.516441
1     3.716080  129.422921  18630.057858     6.635246         NaN
2     8.099124  224.236259  19909.541732     9.275884         NaN
3     8.316766  214.373394  22018.417441     8.059332  356.886136
4     9.092223  181.101509  17978.986339     6.546600  310.135738
...        ...         ...           ...          ...         ...
3271  4.668102  193.681735  47580.991603     7.166639  359.948574
3272  7.808856  193.553212  17329.802160     8.061362         NaN
3273  9.419510  175.762646  33155.578218     7.350233         NaN
3274  5.126763  230.603758  11983.869376     6.303357         NaN
3275  7.874671  195.102299  17404.177061     7.509306         NaN

      Conductivity  Organic_carbon  Trihalomethanes  Turbidity  Potabili
ty
0       564.308654       10.379783        86.990970   2.963135
0
1       592.885359       15.180013        56.329076   4.500656
0
2       418.606213       16.868637        66.420093   3.055934
0
3       363.266516       18.436524       100.341674   4.628771
0
4       398.410813       11.558279        31.997993   4.075075
0
...            ...             ...              ...        ...
...
3271    526.424171       13.894419        66.687695   4.435821
1
3272    392.449580       19.903225              NaN   2.798243
1
3273    432.044783       11.039070        69.845400   3.298875
1
3274    402.883113       11.168946        77.488213   4.708658
1
3275    327.459760       16.140368        78.698446   2.309149
1

[3276 rows x 10 columns]>
```

In [6]:
```python
df.isnull().sum()
```

Out[6]:
```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

In [7]:
```python
df.corr()
```

Out[7]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_ca
ph	1.000000	0.082096	-0.089288	-0.034350	0.018203	0.018614	0.04
Hardness	0.082096	1.000000	-0.046899	-0.030054	-0.106923	-0.023915	0.00
Solids	-0.089288	-0.046899	1.000000	-0.070148	-0.171804	0.013831	0.01
Chloramines	-0.034350	-0.030054	-0.070148	1.000000	0.027244	-0.020486	-0.01
Sulfate	0.018203	-0.106923	-0.171804	0.027244	1.000000	-0.016121	0.03
Conductivity	0.018614	-0.023915	0.013831	-0.020486	-0.016121	1.000000	0.02
Organic_carbon	0.043503	0.003610	0.010242	-0.012653	0.030831	0.020966	1.00
Trihalomethanes	0.003354	-0.013013	-0.009143	0.017084	-0.030274	0.001285	-0.01
Turbidity	-0.039057	-0.014449	0.019546	0.002363	-0.011187	0.005798	-0.02
Potability	-0.003556	-0.013837	0.033743	0.023779	-0.023577	-0.008128	-0.03

In [8]:
```python
df["ph"]=df["ph"].fillna(df["ph"].mean())
df["Sulfate"] = df["Sulfate"].fillna(df["Sulfate"].mean())
df["Trihalomethanes"] = df["Trihalomethanes"].fillna(df["Trihalomethane
s"].mean())
```

In [9]:
```python
df.isna().sum()
```

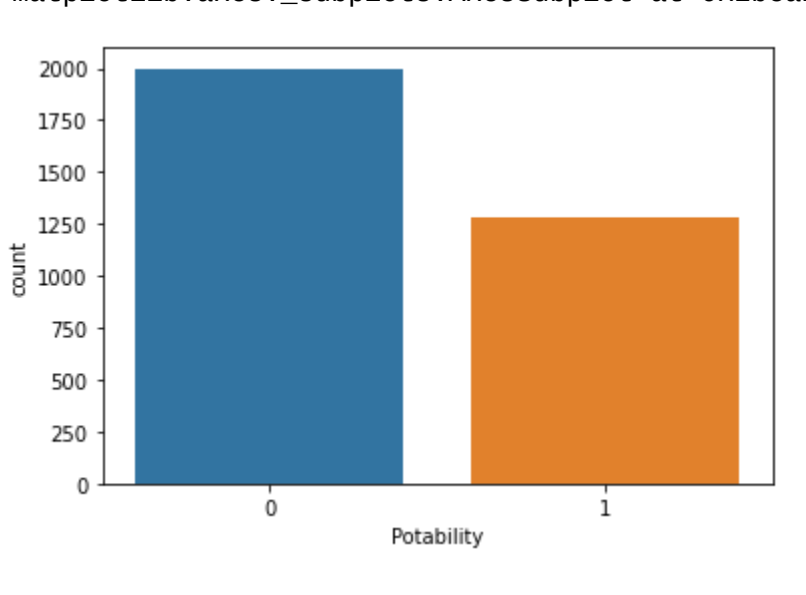
Out[9]:
```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
dtype: int64
```

In [10]:
```python
df.duplicated().sum()
```

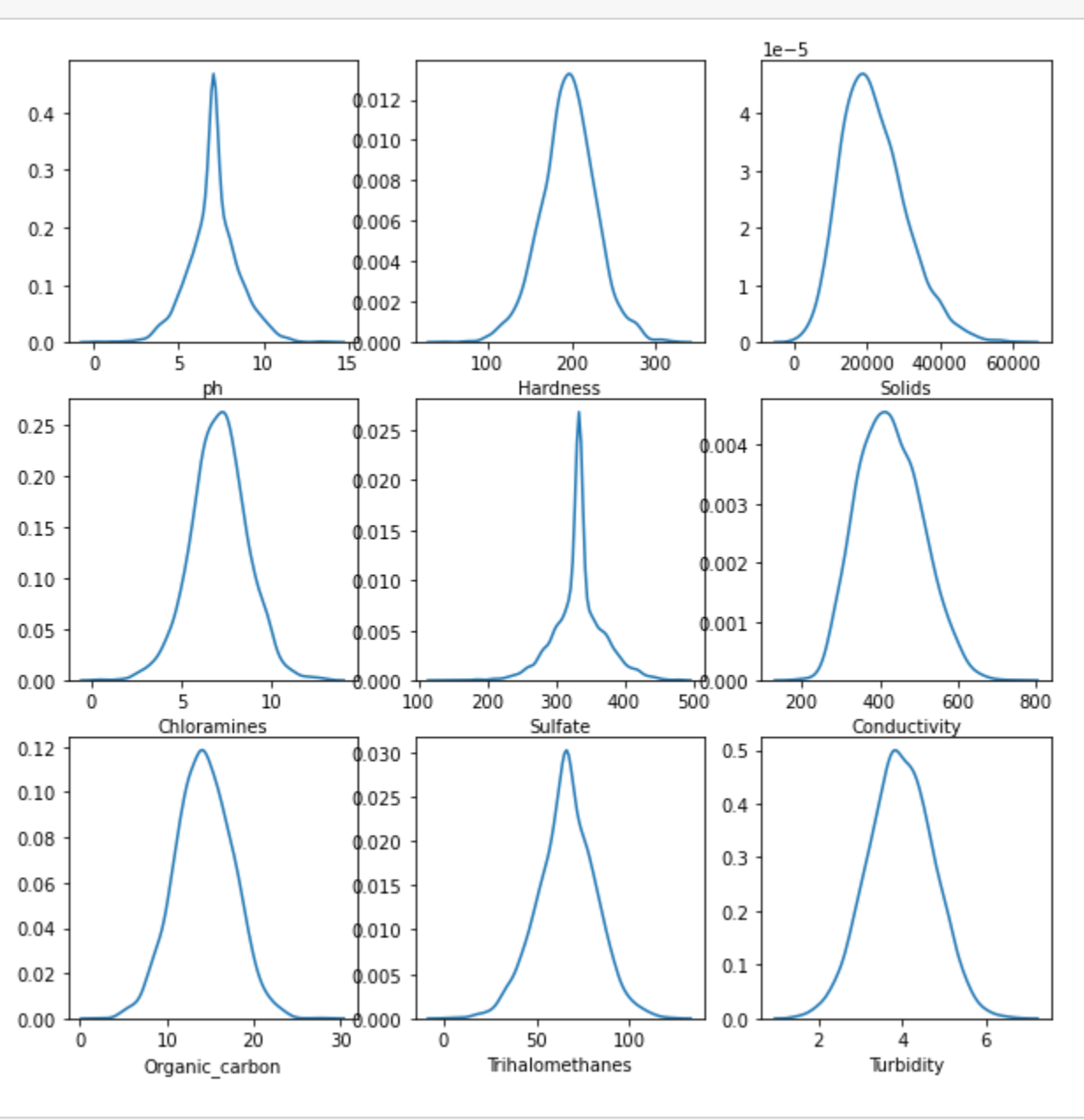
Out[10]: 0

In [12]:
```python
sns.countplot(x=df["Potability"],data=df)
```

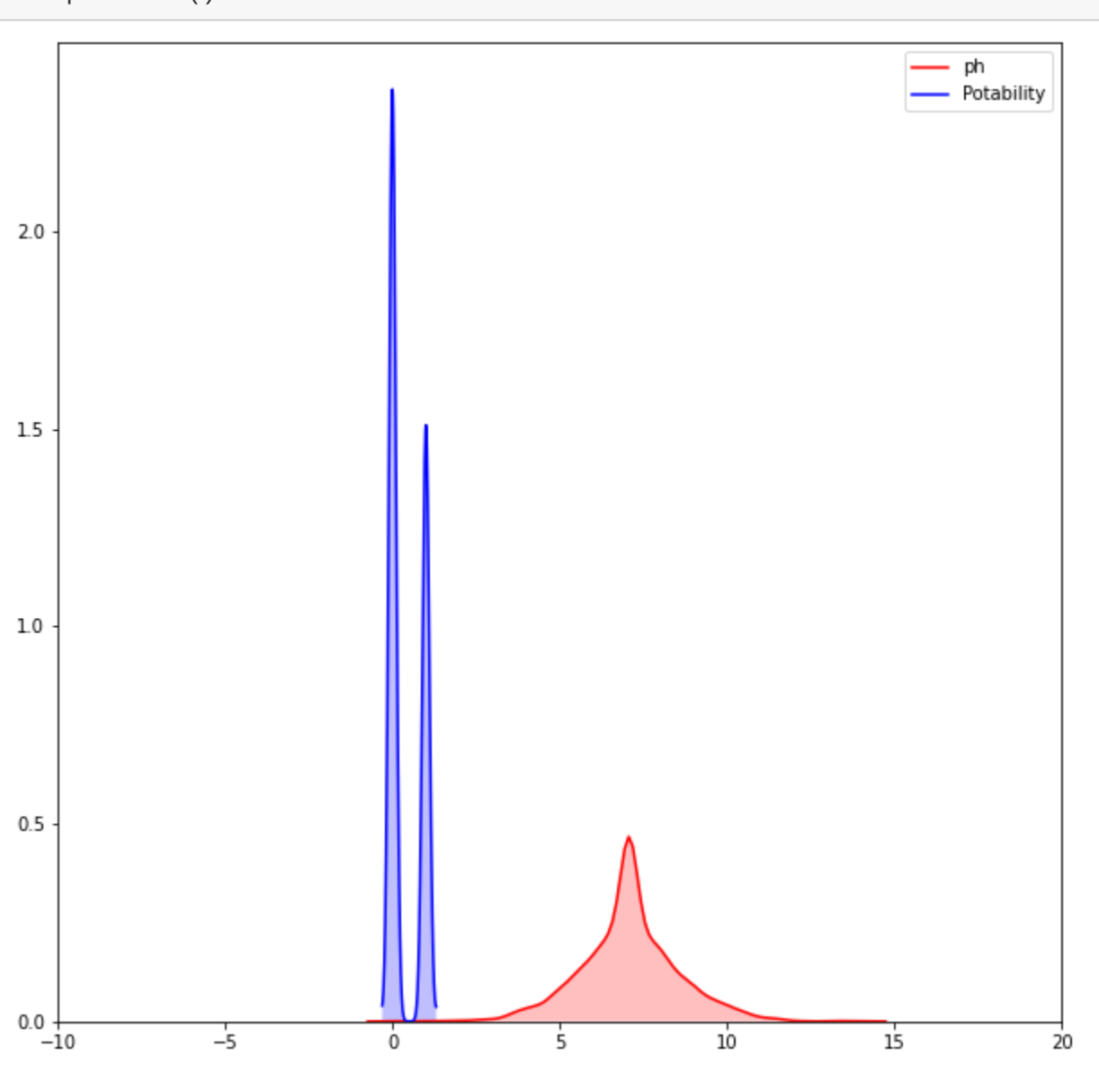
Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x2bea2e1bbe0>

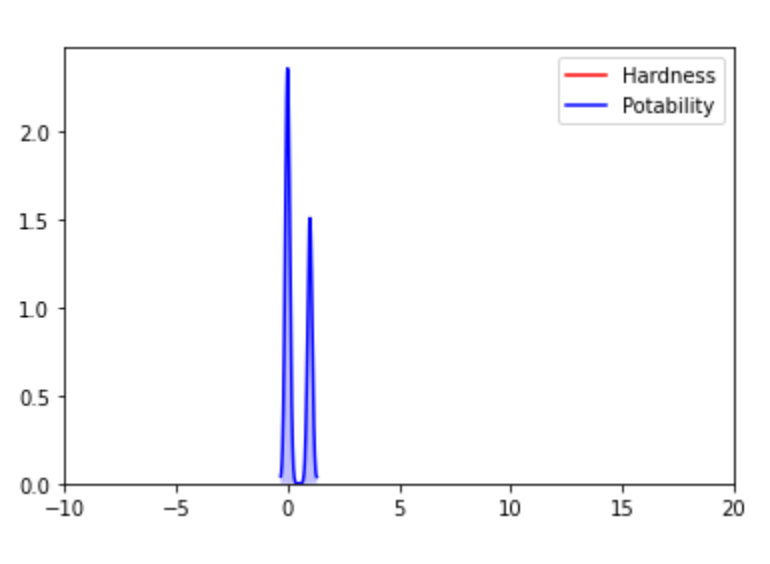


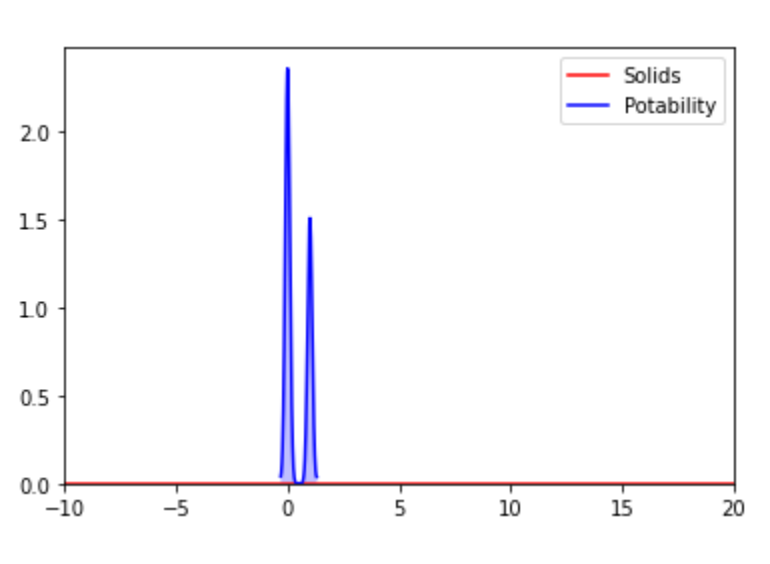
In [14]:
```python
plt.figure(figsize=(10,10))
newdf = df.drop("Potability",axis=True)
for i,column in enumerate(newdf.columns,1):
    plt.subplot(3,3,i)
    sns.distplot(newdf[column],hist=False)
```

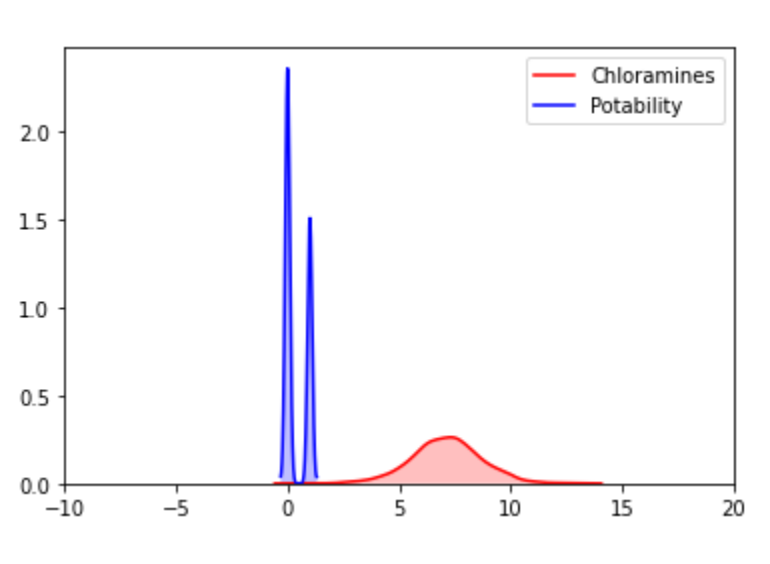


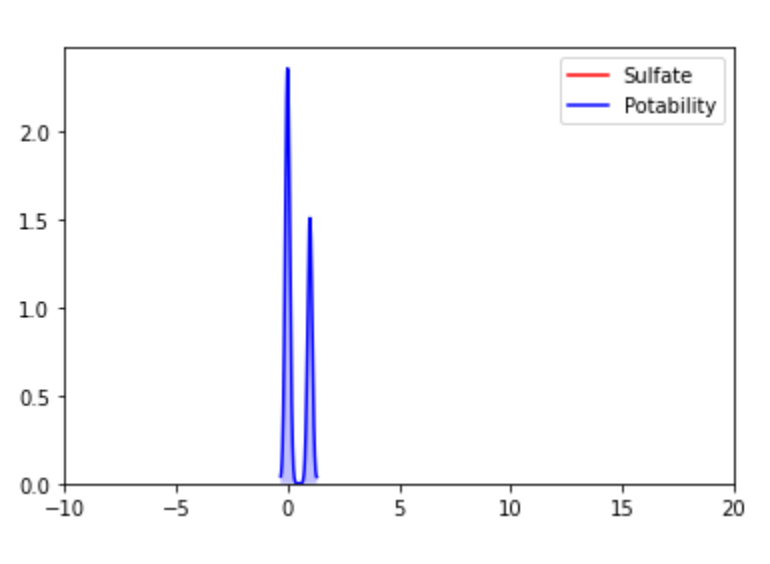
In [24]:
```python
plt.figure(figsize=(10,10))
for i,column in enumerate(newdf.columns,1):
    sns.kdeplot(newdf[column],color="red",shade=True)
    sns.kdeplot(df["Potability"],color="blue",shade=True)
    plt.xlim([-10,20])
    plt.show()
```

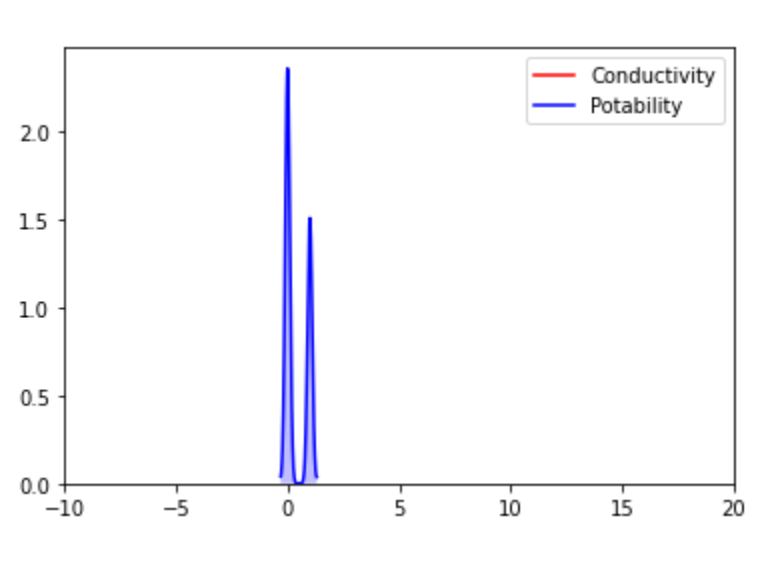


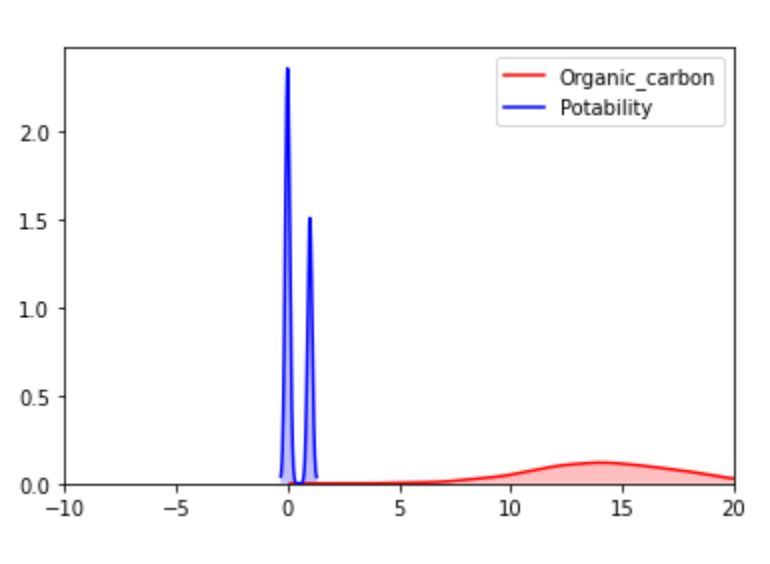


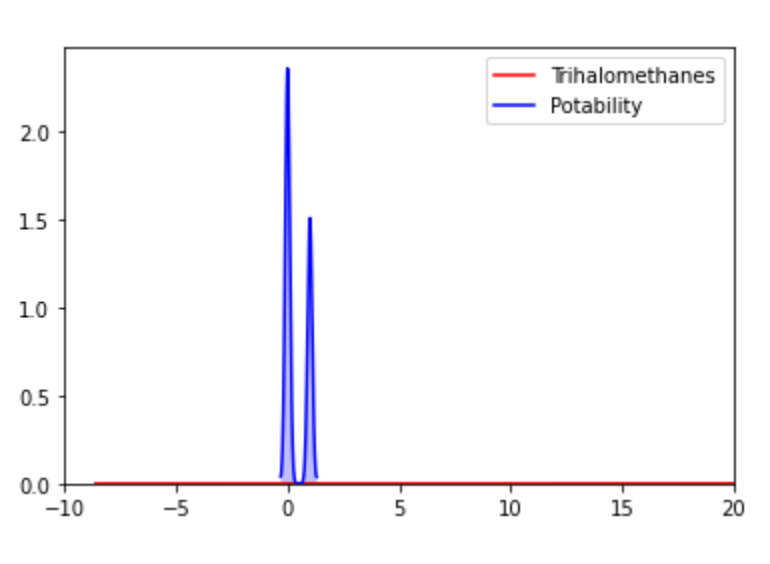


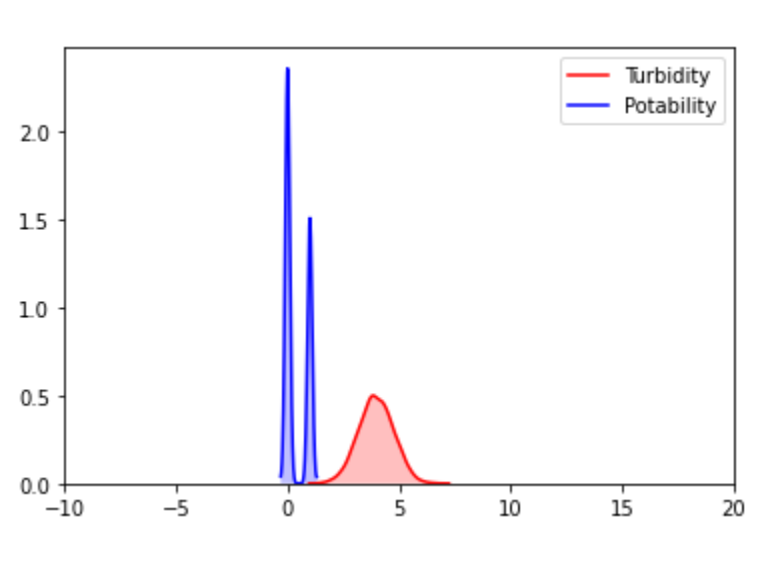






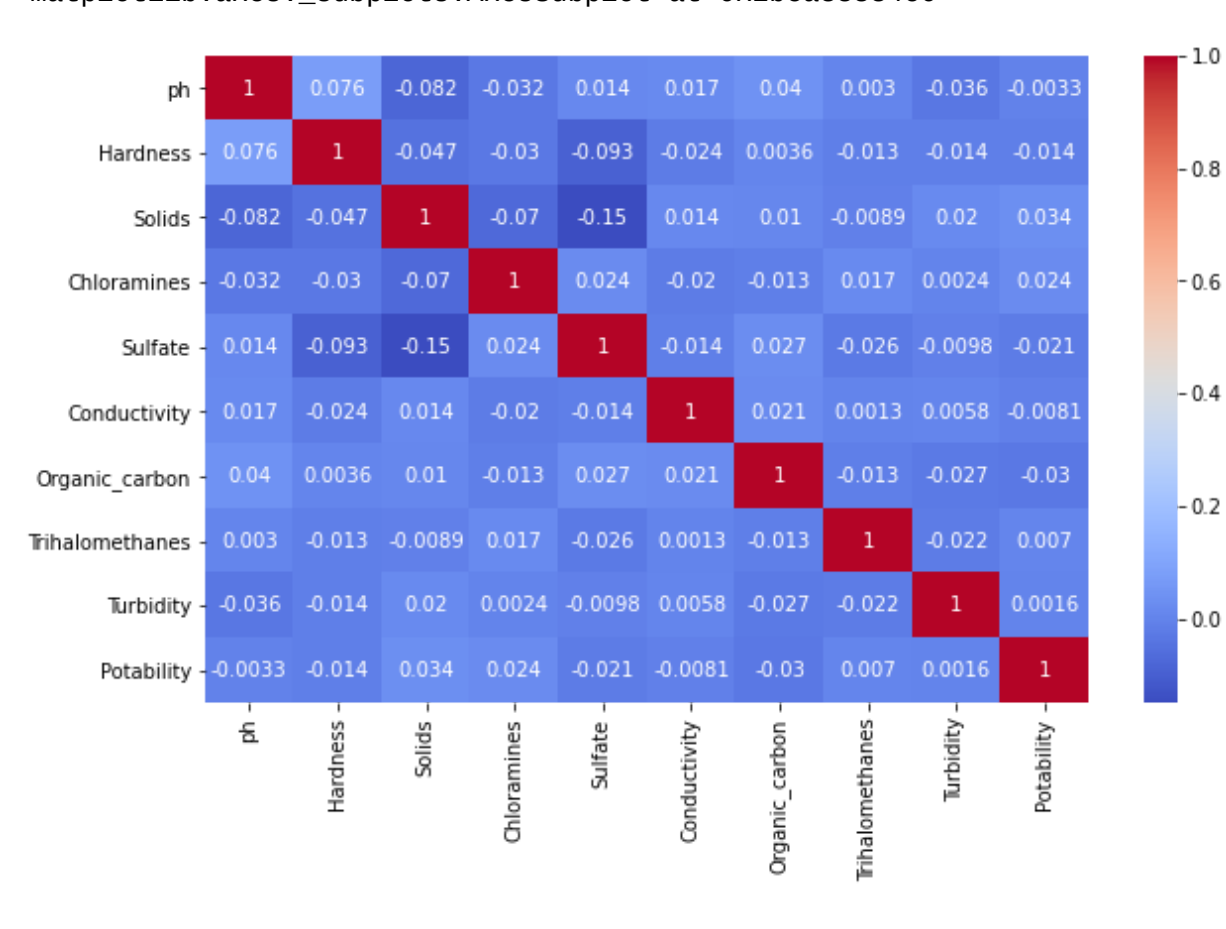






In [27]:
```python
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(),cmap='coolwarm',annot=True)
```

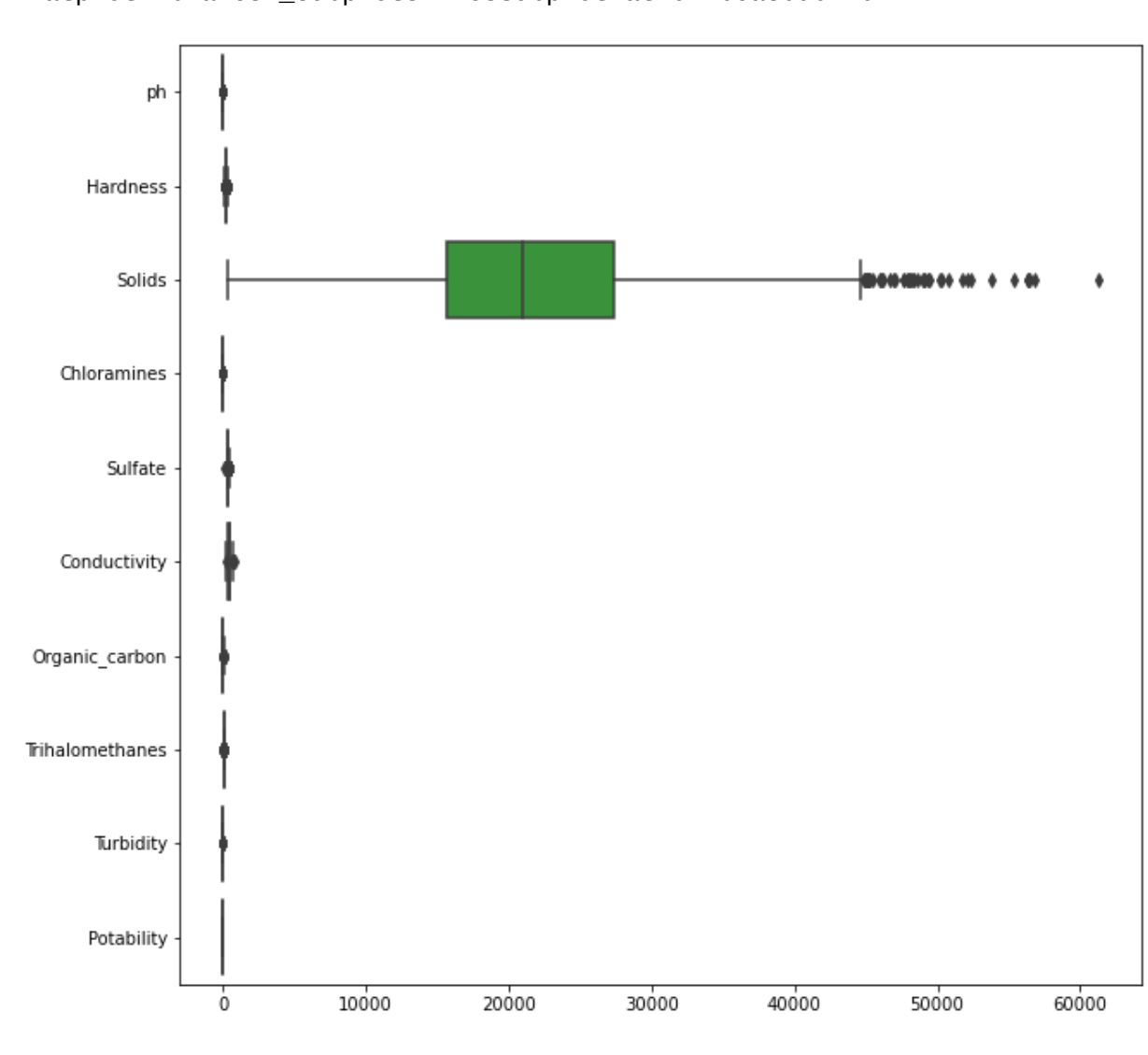
Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x2bea33334c0>



Outlier Treatment

In [29]:
```python
plt.figure(figsize=(10,10))
sns.boxplot(data=df,orient="h")
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x2bea30d07f0>



In [33]:
```python
from scipy import stats
zscore=np.abs(stats.zscore(df))
print(zscore)
```

```
[[8.46038683e-15 2.59194711e-01 1.39470871e-01 ... 1.30614943e+00
  1.28629758e+00 7.99774743e-01]
 [2.28933938e+00 2.03641367e+00 3.85986650e-01 ... 6.38479983e-01
  6.84217891e-01 7.99774743e-01]
 [6.92867789e-01 8.47664833e-01 2.40047337e-01 ... 1.50940884e-03
  1.16736546e+00 7.99774743e-01]
 ...
 [1.59125368e+00 6.26829230e-01 1.27080989e+00 ... 2.18748247e-01
  8.56006782e-01 1.25035206e+00]
 [1.32951593e+00 1.04135450e+00 1.14405809e+00 ... 7.03468419e-01
  9.50797383e-01 1.25035206e+00]
 [5.40150905e-01 3.85462310e-02 5.25811937e-01 ... 7.80223466e-01
  2.12445866e+00 1.25035206e+00]]
```

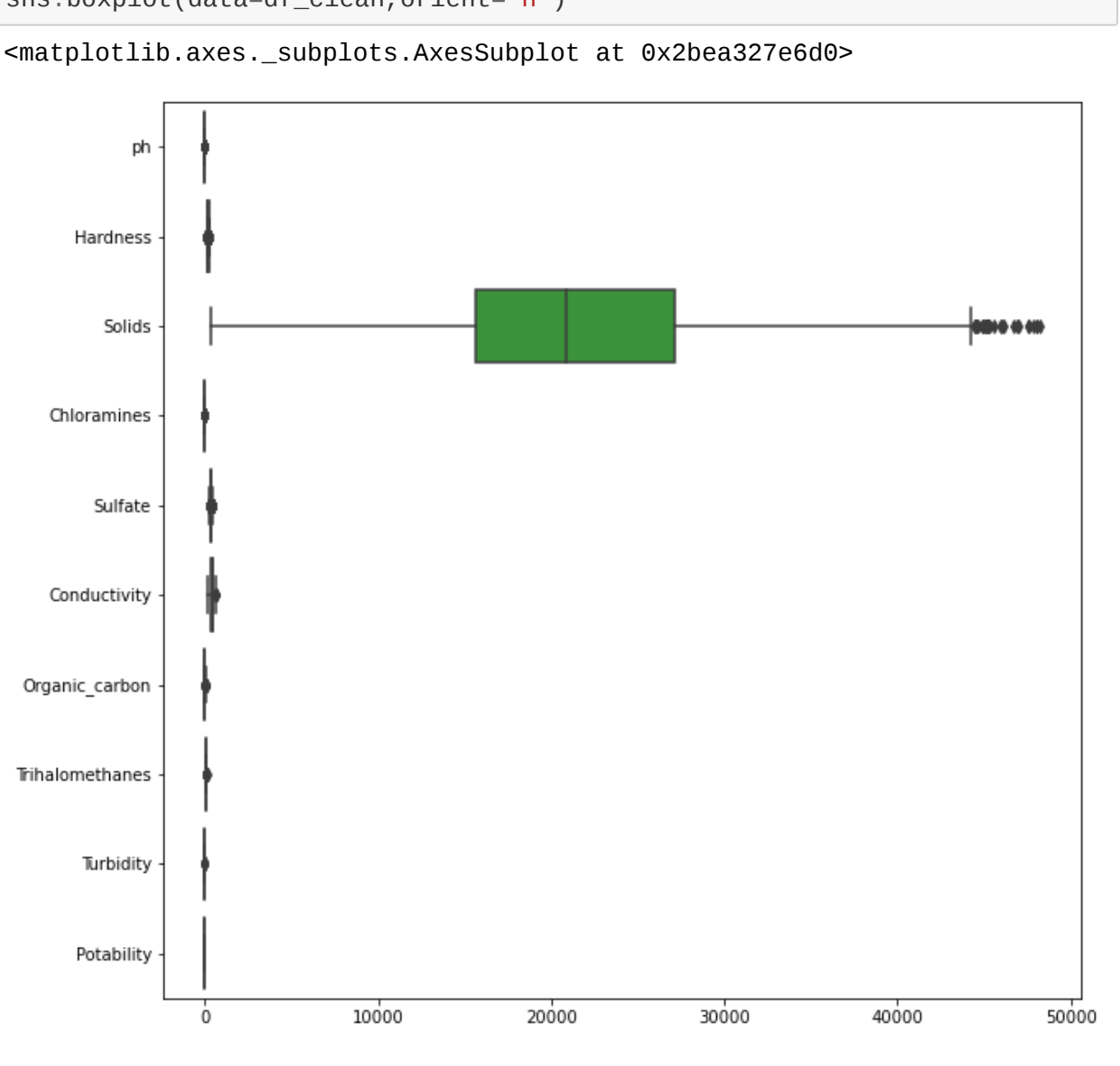
In [34]:
```python
threshold=3
print(np.where(zscore>3))
```

```
(array([  37,   61,   66,   80,   88,  104,  140,  245,  253,  263,  26
3,
        272,  272,  275,  275,  278,  283,  285,  287,  304,  317,  321,
        330,  335,  342,  345,  347,  351,  351,  354,  357,  365,  366,
        379,  382,  383,  385,  434,  492,  509,  531,  546,  666,  680,
        692,  703,  726,  757,  781,  783,  785,  786,  810,  810,  951,
        990, 1031, 1068, 1073, 1075, 1077, 1106, 1123, 1156, 1157, 1186,
       1186, 1290, 1292, 1303, 1316, 1343, 1353, 1360, 1366, 1384, 1412,
       1490, 1523, 1537, 1542, 1554, 1554, 1563, 1605, 1630, 1642, 1743,
       1746, 1766, 1773, 1784, 1792, 1798, 1860, 1892, 1955, 2057, 2075,
       2096, 2121, 2134, 2156, 2189, 2204, 2236, 2302, 2318, 2336, 2343,
       2350, 2353, 2370, 2376, 2401, 2424, 2446, 2447, 2470, 2497, 2602,
       2646, 2646, 2681, 2694, 2699, 2704, 2726, 2757, 2853, 2861, 2868,
       2895, 2899, 2921, 2925, 2928, 2945, 2949, 3014, 3014, 3017, 3020,
       3035, 3042, 3088, 3094, 3130, 3142, 3150, 3162, 3179, 3184, 3190,
       3221, 3226, 3269, 3269], dtype=int64), array([1, 7, 5, 0, 1, 0,
5, 7, 4, 0, 1, 3, 4, 3, 4, 1, 2, 3, 3, 3, 1, 1,
       7, 1, 5, 4, 1, 3, 4, 1, 4, 4, 2, 1, 8, 1, 4, 3, 8, 3, 7, 2, 2, 4,
       0, 4, 0, 3, 4, 0, 6, 4, 0, 4, 7, 8, 2, 2, 8, 7, 2, 4, 7, 7, 7, 2,
       4, 8, 4, 0, 7, 0, 0, 7, 4, 5, 4, 1, 4, 4, 1, 2, 4, 4, 4, 7, 4, 4,
       2, 4, 4, 2, 6, 4, 4, 8, 2, 6, 0, 0, 7, 5, 4, 0, 4, 6, 3, 4, 3, 0,
       3, 7, 3, 7, 3, 3, 3, 3, 3, 2, 2, 0, 1, 0, 3, 3, 5, 4, 8, 4, 1, 0,
       0, 0, 8, 0, 0, 0, 7, 0, 2, 0, 0, 7, 8, 0, 0, 2, 5, 2, 2, 1, 7, 2,
       1, 2, 0, 1], dtype=int64))
```

In [35]:
```python
df_clean=df
df_clean = df_clean[(zscore<3).all(axis=1)]
```

In [38]:
```python
plt.figure(figsize=(10,10))
sns.boxplot(data=df_clean,orient="h")
```

Out[38]: <matplotlib.axes._subplots.AxesSubplot at 0x2bea327e6d0>



In [40]:
```python
df.shape,df_clean.shape
```

Out[40]: ((3276, 10), (3128, 10))

In [41]:
```python
X=df.drop(["Potability"],axis=1)
y=df["Potability"]
```

In [43]:
```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_
state=1)
```

Normalizing and Scaling

In [44]:
```python
from sklearn.preprocessing import MinMaxScaler
mscale=MinMaxScaler()
mscale.fit_transform(x_train)
mscale.transform(x_test)
```

Out[44]:
```
array([[0.43309677, 0.41109394, 0.23731887, ..., 0.38731862, 0.55410795,
        0.38142124],
       [0.50577104, 0.67317199, 0.43920245, ..., 0.55819713, 0.41007286,
        0.5359952 ],
       [0.66091832, 0.43747643, 0.82024943, ..., 0.37355762, 0.4990904 ,
        0.34989333],
       ...,
       [0.42761385, 0.82865848, 0.34328397, ..., 0.36717943, 0.74223794,
        0.68094042],
       [0.50577104, 0.63943978, 0.25568363, ..., 0.49939759, 0.38501381,
        0.5119923 ],
       [0.54300868, 0.55105395, 0.18101919, ..., 0.6025204 , 0.50387526,
        0.42519848]])
```

Train Models

In [45]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,confusion_matrix
```

In [46]:
```python
keys = ['KNN','LogisticRegression','DecisionTree','Randomforest','suppor
tvector','Gaussian']
values = [KNeighborsClassifier(),LogisticRegression(),DecisionTreeClassi
fier(),RandomForestClassifier(),SVC(),GaussianNB()]
models = dict(zip(keys,values))
print(models)
```

```
{'KNN': KNeighborsClassifier(), 'LogisticRegression': LogisticRegression
(), 'DecisionTree': DecisionTreeClassifier(), 'Randomforest': RandomFore
stClassifier(), 'supportvector': SVC(), 'Gaussian': GaussianNB()}
```

In [47]:
```python
for name,algo in models.items():
    model = algo
    model.fit(x_train,y_train)
    predict = model.predict(x_test)
    acc = model.score(x_train,y_train)
    print(name,acc)
```

```
KNN 0.6950381679389313
LogisticRegression 0.6206106870229008
DecisionTree 1.0
Randomforest 1.0
supportvector 0.6202290076335878
Gaussian 0.6347328244274809
```

Testing

In [48]:
```python
clf = RandomForestClassifier()
clf.fit(x_train,y_train)
test_score = clf.score(x_test,y_test)
y_pred=clf.predict(x_test)
print("RandomForestClassifier Test Score:",test_score)
```

```
RandomForestClassifier Test Score: 0.6371951219512195
```

In [ ]: