Water Potability Prediction

In [1]:
```
# Importing required packages:

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
# Invoke data from CSV file:

waterpot_dataset=pd.read_csv('water_potability.csv')
print(waterpot_dataset)
```

```
        Hardness        Solids  Chloramines     Sulfate  Conductivity  \
0     204.890456  20791.31898     7.300212  368.516441    564.308654
1     129.422921  18630.05786     6.635246         NaN    592.885359
2     224.236259  19909.54173     9.275884         NaN    418.606213
3     214.373394  22018.41744     8.059332  356.886136    363.266516
4     181.101509  17978.98634     6.546600  310.135738    398.410813
...          ...          ...          ...         ...           ...
3271  193.681736  47580.99160     7.166639  359.948574    526.424171
3272  193.553212  17329.80216     8.061362         NaN    392.449580
3273  175.762646  33155.57822     7.350233         NaN    432.044783
3274  230.603758  11983.86938     6.303357         NaN    402.883113
3275  195.102299  17404.17706     7.509306         NaN    327.459761

      Organic_carbon  Trihalomethanes  Turbidity  Potability        ph
0          10.379783        86.990970   2.963135           0       NaN
1          15.180013        56.329076   4.500656           0  3.716080
2          16.868637        66.420093   3.055934           0  8.099124
3          18.436525       100.341674   4.628771           0  8.316766
4          11.558279        31.997993   4.075075           0  9.092223
...              ...              ...        ...         ...       ...
3271       13.894419        66.687695   4.435821           1  4.668102
3272       19.903225              NaN   2.798243           1  7.808856
3273       11.039070        69.845400   3.298875           1  9.419510
3274       11.168946        77.488213   4.708658           1  5.126763
3275       16.140368        78.698446   2.309149           1  7.874671

[3276 rows x 10 columns]
```

In [3]:
```
# Printing the first 5 rows from data set:

waterpot_dataset.head()
```

Out[3]:

	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability	ph
0	204.890456	20791.31898	7.300212	368.516441	564.308654	10.379783	86.990970	2.963135	0	NaN
1	129.422921	18630.05786	6.635246	NaN	592.885359	15.180013	56.329076	4.500656	0	3.716080
2	224.236259	19909.54173	9.275884	NaN	418.606213	16.868637	66.420093	3.055934	0	8.099124
3	214.373394	22018.41744	8.059332	356.886136	363.266516	18.436525	100.341674	4.628771	0	8.316766
4	181.101509	17978.98634	6.546600	310.135738	398.410813	11.558279	31.997993	4.075075	0	9.092223

In [4]:
```
# Shape of the data:

waterpot_dataset.shape
```

Out[4]: (3276, 10)

In [5]:
```
# Getting information about the data:

waterpot_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3276 entries, 0 to 3275
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Hardness         3276 non-null   float64
 1   Solids           3276 non-null   float64
 2   Chloramines      3276 non-null   float64
 3   Sulfate          2495 non-null   float64
 4   Conductivity     3276 non-null   float64
 5   Organic_carbon   3276 non-null   float64
 6   Trihalomethanes  3114 non-null   float64
 7   Turbidity        3276 non-null   float64
 8   Potability       3276 non-null   int64
 9   ph               2785 non-null   float64
dtypes: float64(9), int64(1)
memory usage: 256.0 KB
```

In [6]:
```
# Statistical measures about the data:

waterpot_dataset.describe()
```

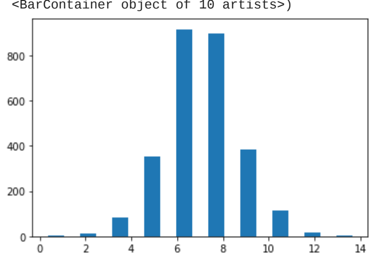
Out[6]:

	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability	ph
count	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.000000	3114.000000	3276.000000	3276.000000	2785.000000
mean	196.369496	22014.092526	7.122277	333.775777	426.205111	14.284970	66.396293	3.966786	0.390110	7.080795
std	32.879761	8768.570828	1.583085	41.416840	80.824064	3.308162	16.175008	0.780382	0.487849	1.594320
min	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	0.738000	1.450000	0.000000	0.000000
25%	176.850538	15666.690300	6.127421	307.699498	365.734414	12.065801	55.844536	3.439711	0.000000	6.093092
50%	196.967627	20927.833605	7.130299	333.073546	421.884968	14.218338	66.622485	3.955028	0.000000	7.036752
75%	216.667456	27332.762125	8.114887	359.950170	481.792305	16.557652	77.337473	4.500320	1.000000	8.062066
max	323.124000	61227.196010	13.127000	481.030642	753.342620	28.300000	124.000000	6.739000	1.000000	14.000000

Data Visualization

In [7]:
```
# Plotting the ph value:

plt.hist(waterpot_dataset['ph'],rwidth = 0.5)
```

Out[7]:
```
(array([  4.,  12.,  84., 353., 915., 898., 382., 116.,  17.,   4.]),
 array([ 0. ,  1.4,  2.8,  4.2,  5.6,  7. ,  8.4,  9.8, 11.2, 12.6, 14. ]),
 <BarContainer object of 10 artists>)
```

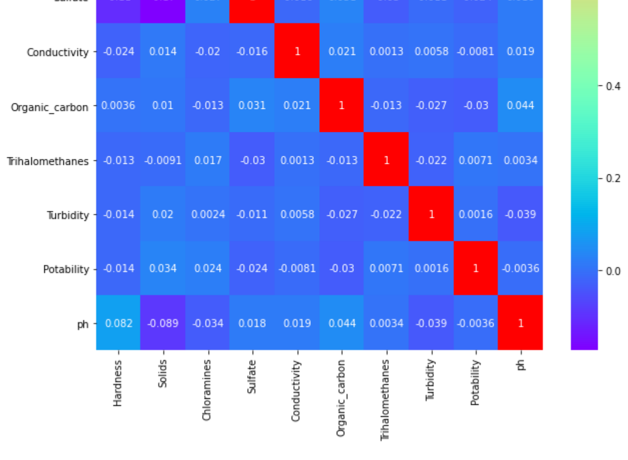


In [8]:
```
# plotting the potability:

plt.hist(waterpot_dataset['Potability'],rwidth = 0.5)
```

Out[8]:
```
(array([1998.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,    0.,
        1278.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <BarContainer object of 10 artists>)
```

In [9]:
```
# Constructing a heatmap:

waterpot = waterpot_dataset.corr()
plt.figure(figsize = (10,10))
sns.heatmap(waterpot,cbar = True,annot=True,cmap ='rainbow')
```

Out[9]: <AxesSubplot:>



Preprocessing

In [10]:
```
# Separatig features and target:

X = waterpot_dataset.iloc[:,0:3].values
y = waterpot_dataset.iloc[:,2].values
```

In [11]:
```
print(X)
```

```
[[2.04890456e+02 2.07913190e+04 7.30021187e+00]
 [1.29422921e+02 1.86300579e+04 6.63524588e+00]
 [2.24236259e+02 1.99095417e+04 9.27588360e+00]
 ...
 [1.75762646e+02 3.31555782e+04 7.35023323e+00]
 [2.30603758e+02 1.19838694e+04 6.30335653e+00]
 [1.95102299e+02 1.74041771e+04 7.50930586e+00]]
```

In [12]:
```
print(y)
```

```
[7.30021187 6.63524588 9.2758836  ... 7.35023323 6.30335653 7.50930586]
```

In [13]:
```
# Splitting the data into train and test data:

from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.35)
```

In [14]:
```
# Shape of X,X_train and X_test:

print(X.shape,X_train.shape,X_test.shape)
```

```
(3276, 3) (2129, 3) (1147, 3)
```

Using Multiple Linear Regressor:

In [15]:
```
# Loading the model:

from sklearn.linear_model import LinearRegression
reg=LinearRegression()
```

In [16]:
```
# Fitting the trained data:

reg.fit(X_train,y_train)
```

Out[16]: LinearRegression()

In [17]:
```
# Prediction on test data:

y_pre = reg.predict(X_test)
```

In [18]:
```
# Score for Multiple Linear Regressor:

from sklearn.metrics import r2_score
result=r2_score(y_test,y_pre)
print(result*100)
```

```
100.0
```

In [19]:
```
# Real value vs predicted value (visualization):

y_test=list(y_test)
plt.plot(y_test,color='blue',label='ph')
plt.plot(y_pre,color='red',label='Predicted ph')
plt.legend()
plt.show()
```

Using Random Forest Regressor:

In [20]:
```
# Loading the model:

from sklearn.ensemble import RandomForestRegressor
reg=RandomForestRegressor(n_estimators=100)
```

In [21]:
```
# Fitting the trained data:

reg.fit(X_train,y_train)
```

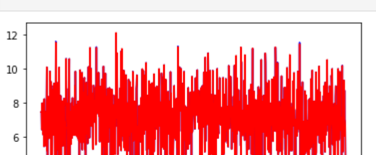
Out[21]: RandomForestRegressor()

In [22]:
```
# Prediction on test data:

y_pre = reg.predict(X_test)
```

In [23]:
```
# Score for Random Forest Regressor:

from sklearn.metrics import r2_score
result=r2_score(y_test,y_pre)
print(result*100)
```

```
99.99183070420156
```

In [24]:
```
# Real value vs predicted value (visualization):

y_test=list(y_test)
plt.plot(y_test,color='blue',label='ph')
plt.plot(y_pre,color='red',label='Predicted ph')
plt.legend()
plt.show()
```



Using Support Vector Regressor:

In [25]:
```
# Loading the model:

from sklearn.svm import SVR
reg=SVR()
```

In [26]:
```
# Fitting the trained data:

reg.fit(X_train,y_train)
```

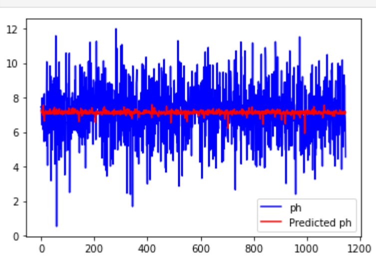
Out[26]: SVR()

In [27]:
```
# Prediction on test data:

y_pre = reg.predict(X_test)
```

In [28]:
```
# Score for Support Vector Regressor:

from sklearn.metrics import r2_score
result=r2_score(y_test,y_pre)
print(result*100)
```

```
0.03624377340536444
```

In [29]:
```
# Real value vs predicted value (visualization):

y_test=list(y_test)
plt.plot(y_test,color='blue',label='ph')
plt.plot(y_pre,color='red',label='Predicted ph')
plt.legend()
plt.show()
```



Using K-Nearest Neighbors Regressor

In [30]:
```
# Loading the model:

from sklearn.neighbors import KNeighborsRegressor
reg = KNeighborsRegressor(n_neighbors=3)
```

In [31]:
```
# Fitting the trained data:

reg.fit(X_train,y_train)
```

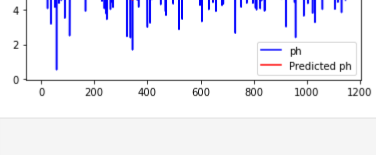
Out[31]: KNeighborsRegressor(n_neighbors=3)

In [32]:
```
# Prediction on test data:

y_pre = reg.predict(X_test)
```

In [33]:
```
# Score for K-Nearest neighbors Regressor:

from sklearn.metrics import r2_score
result=r2_score(y_test,y_pre)
print(result*100)
```

```
-32.94555838696538
```

In [34]:
```
# Real value vs predicted value (visualization):

y_test=list(y_test)
plt.plot(y_test,color='blue',label='ph')
plt.plot(y_pre,color='red',label='Predicted ph')
plt.legend()
plt.show()
```



In [ ]:

In [35]:
```
from sklearn.cluster import KMeans
kmn=KMeans(n_clusters=10)
```

In [36]:
```
# Fitting the trained data:

reg.fit(X_train,y_train)
```

Out[36]: KNeighborsRegressor(n_neighbors=3)

In [37]:
```
# Prediction on test data:

y_pre = reg.predict(X_test)
```

In [38]:
```
# Score for K-Nearest neighbors Regressor:

from sklearn.metrics import r2_score
result=r2_score(y_test,y_pre)
print(result*100)
```

```
-32.94555838696538
```

In [39]:
```
# Real value vs predicted value (visualization):

y_test=list(y_test)
plt.plot(y_test,color='blue',label='ph')
plt.plot(y_pre,color='red',label='Predicted ph')
plt.legend()
plt.show()
```


In [ ]: