RiverSafe

May 11, 2023

[46]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import plotly.express as px
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score,␣
↪classification_report,confusion_matrix
```

[47]:
```python
df=pd.read_csv('F:\\NEW FOLDER\\ml\\project\\water_potability.csv')
df.head(10)
```

[47]:
```
          ph    Hardness        Solids  Chloramines     Sulfate  Conductivity  \
0        NaN  204.890455  20791.318981     7.300212  368.516441    564.308654
1   3.716080  129.422921  18630.057858     6.635246         NaN    592.885359
2   8.099124  224.236259  19909.541732     9.275884         NaN    418.606213
3   8.316766  214.373394  22018.417441     8.059332  356.886136    363.266516
4   9.092223  181.101509  17978.986339     6.546600  310.135738    398.410813
5   5.584087  188.313324  28748.687739     7.544869  326.678363    280.467916
6  10.223862  248.071735  28749.716544     7.513408  393.663396    283.651634
7   8.635849  203.361523  13672.091764     4.563009  303.309771    474.607645
8        NaN  118.988579  14285.583854     7.804174  268.646941    389.375566
9  11.180284  227.231469  25484.508491     9.077200  404.041635    563.885481

   Organic_carbon  Trihalomethanes  Turbidity  Potability
0       10.379783        86.990970   2.963135           0
1       15.180013        56.329076   4.500656           0
2       16.868637        66.420093   3.055934           0
3       18.436524       100.341674   4.628771           0
4       11.558279        31.997993   4.075075           0
5        8.399735        54.917862   2.559708           0
6       13.789695        84.603556   2.672989           0
7       12.363817        62.798309   4.401425           0
8       12.706049        53.928846   3.595017           0
9       17.927806        71.976601   4.370562           0
```

[48]:
```
df.describe()
```

[48]:
```
                ph     Hardness        Solids  Chloramines      Sulfate  \
count  2785.000000  3276.000000   3276.000000  3276.000000  2495.000000
mean      7.080795   196.369496  22014.092526     7.122277   333.775777
std       1.594320    32.879761   8768.570828     1.583085    41.416840
min       0.000000    47.432000    320.942611     0.352000   129.000000
25%       6.093092   176.850538  15666.690297     6.127421   307.699498
50%       7.036752   196.967627  20927.833607     7.130299   333.073546
75%       8.062066   216.667456  27332.762127     8.114887   359.950170
max      14.000000   323.124000  61227.196008    13.127000   481.030642

       Conductivity  Organic_carbon  Trihalomethanes   Turbidity  Potability
count   3276.000000     3276.000000      3114.000000  3276.000000  3276.000000
mean     426.205111       14.284970        66.396293     3.966786     0.390110
std       80.824064        3.308162        16.175008     0.780382     0.487849
min      181.483754        2.200000         0.738000     1.450000     0.000000
25%      365.734414       12.065801        55.844536     3.439711     0.000000
50%      421.884968       14.218338        66.622485     3.955028     0.000000
75%      481.792304       16.557652        77.337473     4.500320     1.000000
max      753.342620       28.300000       124.000000     6.739000     1.000000
```

[49]:
```
df.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
```

for finding the null values

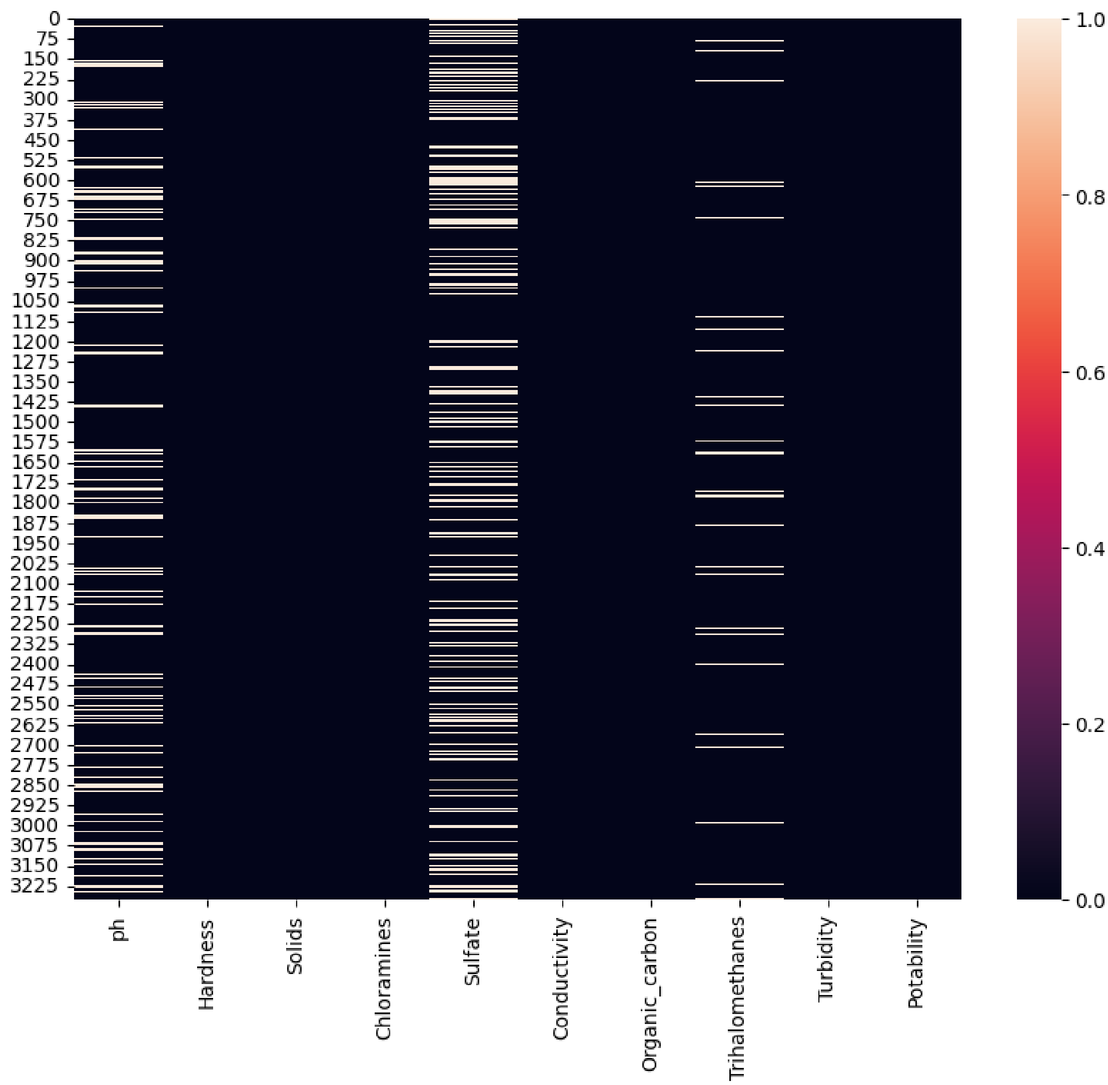
[50]:
```
df.isnull().sum()
```

[50]:
```
ph                 491
Hardness             0
```

```
Solids                0
Chloramines           0
Sulfate             781
Conductivity          0
Organic_carbon        0
Trihalomethanes     162
Turbidity             0
Potability            0
dtype: int64
```

[51]:
```
plt.figure(figsize=(10,8))
sns.heatmap(df.isnull())
```

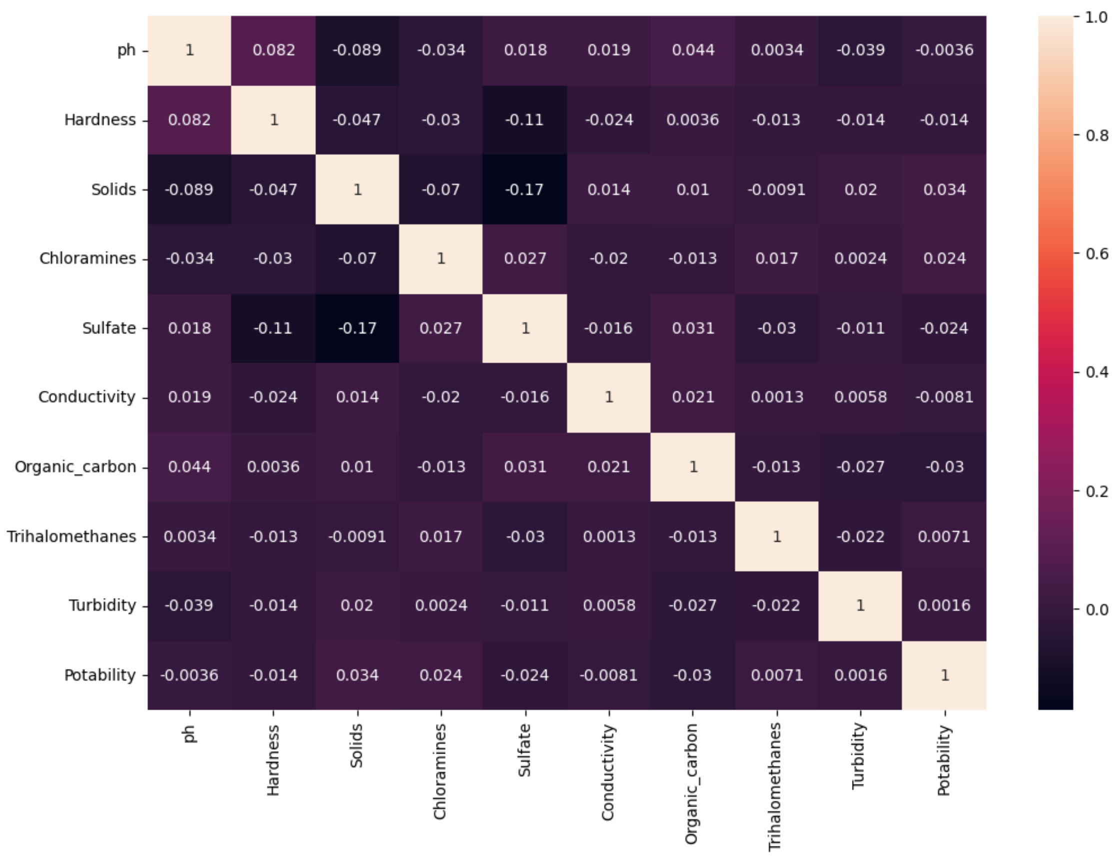
[51]: <Axes: >



1 for finding the correlations

[52]:
```
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(),annot=True)
```

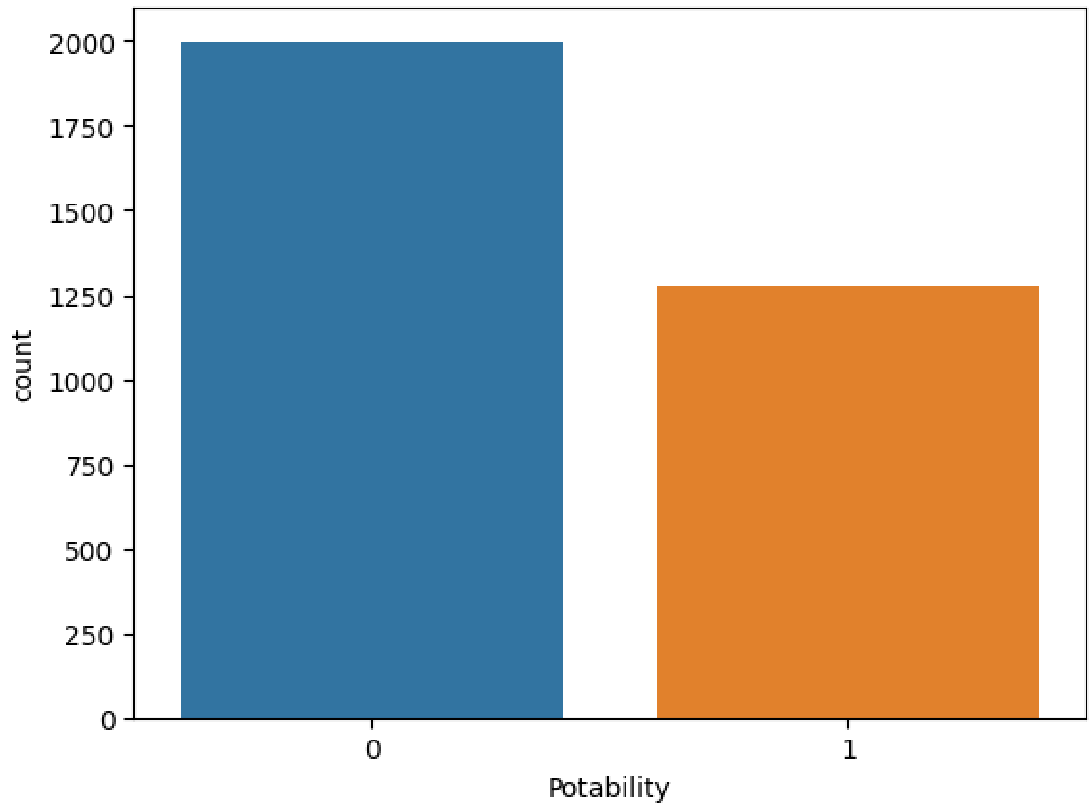
[52]: <Axes: >



2 for finding the count of the different types of values in target coloumn

[53]:
```
sns.countplot(x='Potability',data=df)
```

[53]: <Axes: xlabel='Potability', ylabel='count'>

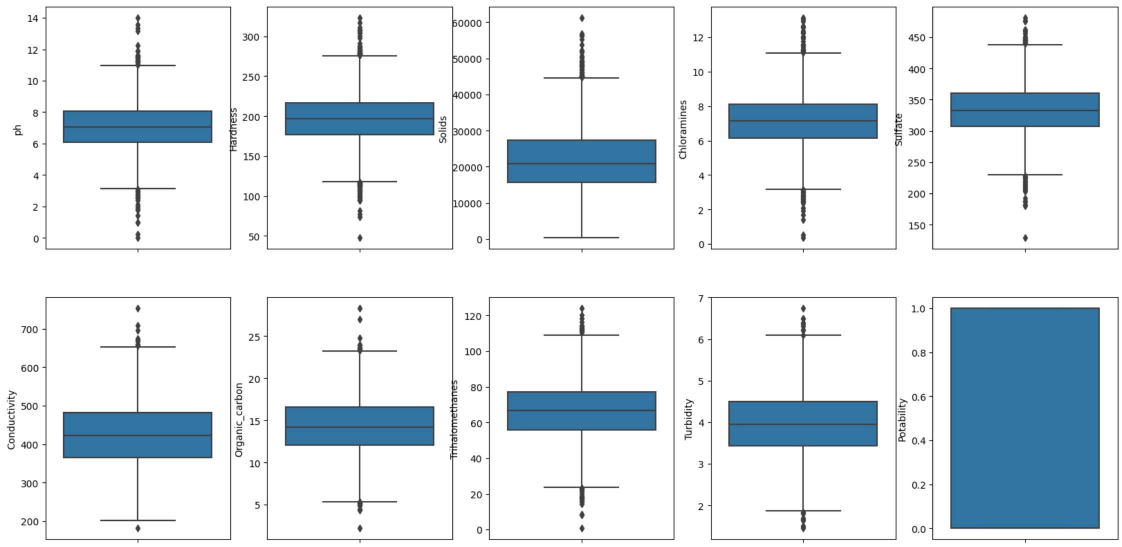


[54]:
```
df['Potability'].value_counts()
```

[54]:
```
0    1998
1    1278
Name: Potability, dtype: int64
```

3 finding outliers

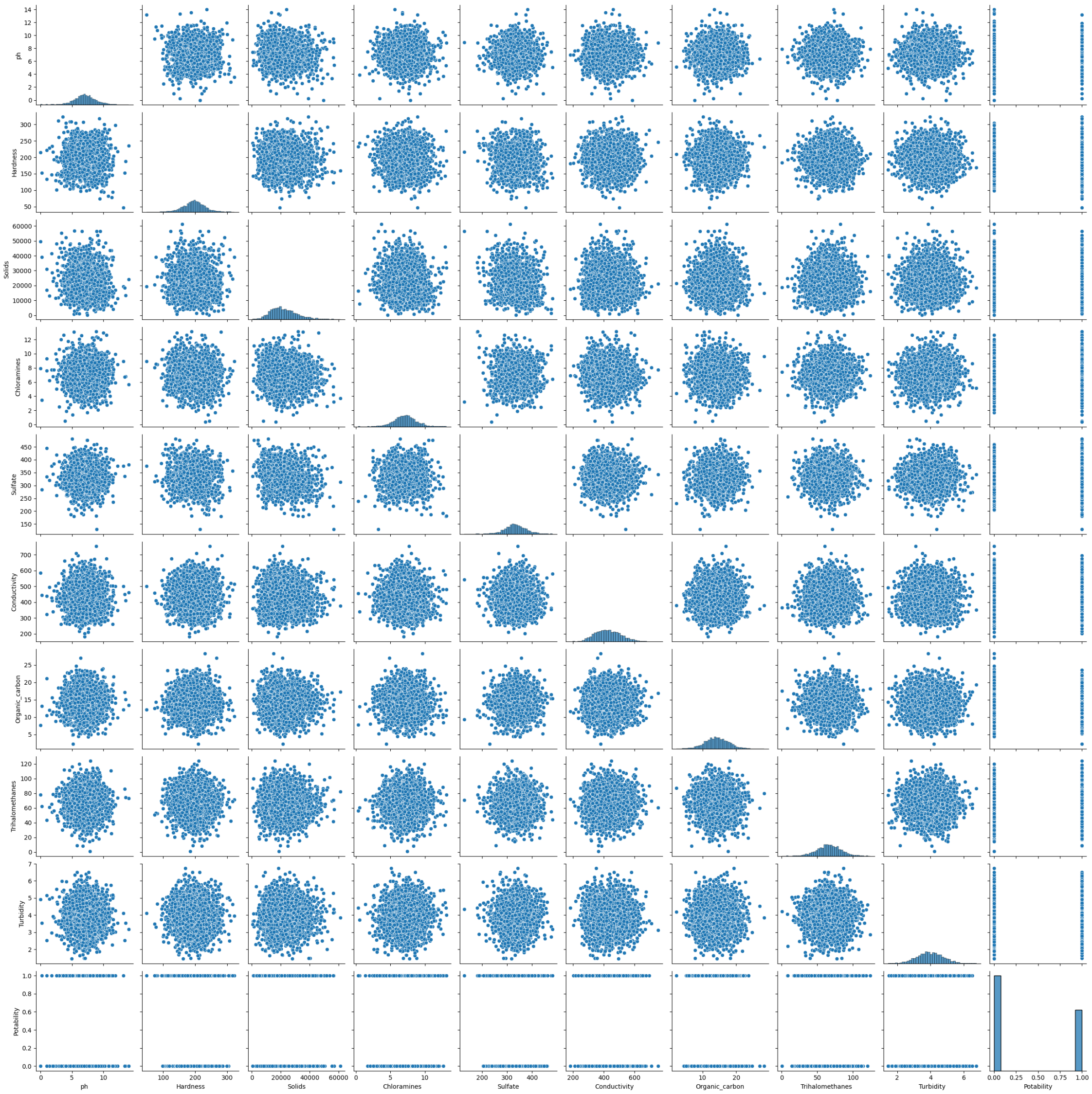
[55]:
```
fig,ax=plt.subplots(ncols=5,nrows=2,figsize=(20,10))
ax=ax.flatten()
index=0
for col,values in df.items():
    sns.boxplot(y=col,data=df,ax=ax[index])
    index +=1
```



4 detailed correlation(more scattered means less correlation)

[56]:
```
sns.pairplot(df)
```

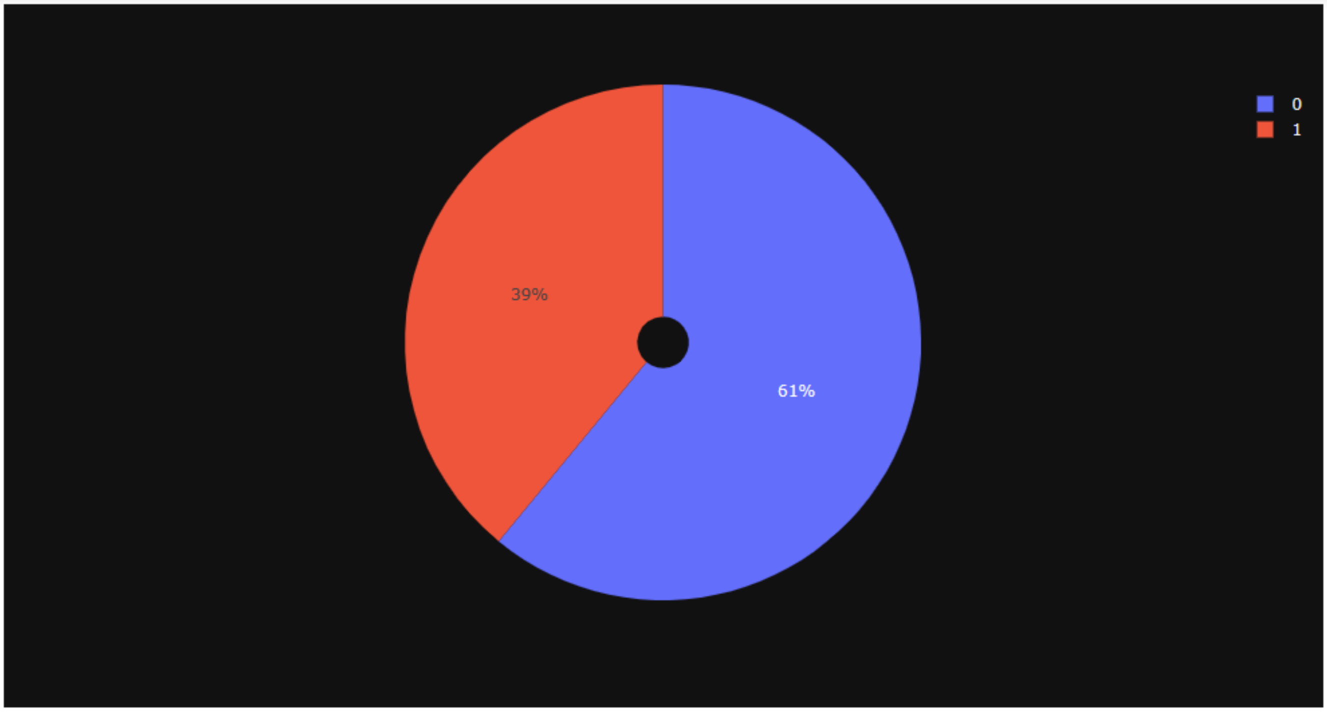
[56]: `<seaborn.axisgrid.PairGrid at 0x1f052bfb460>`



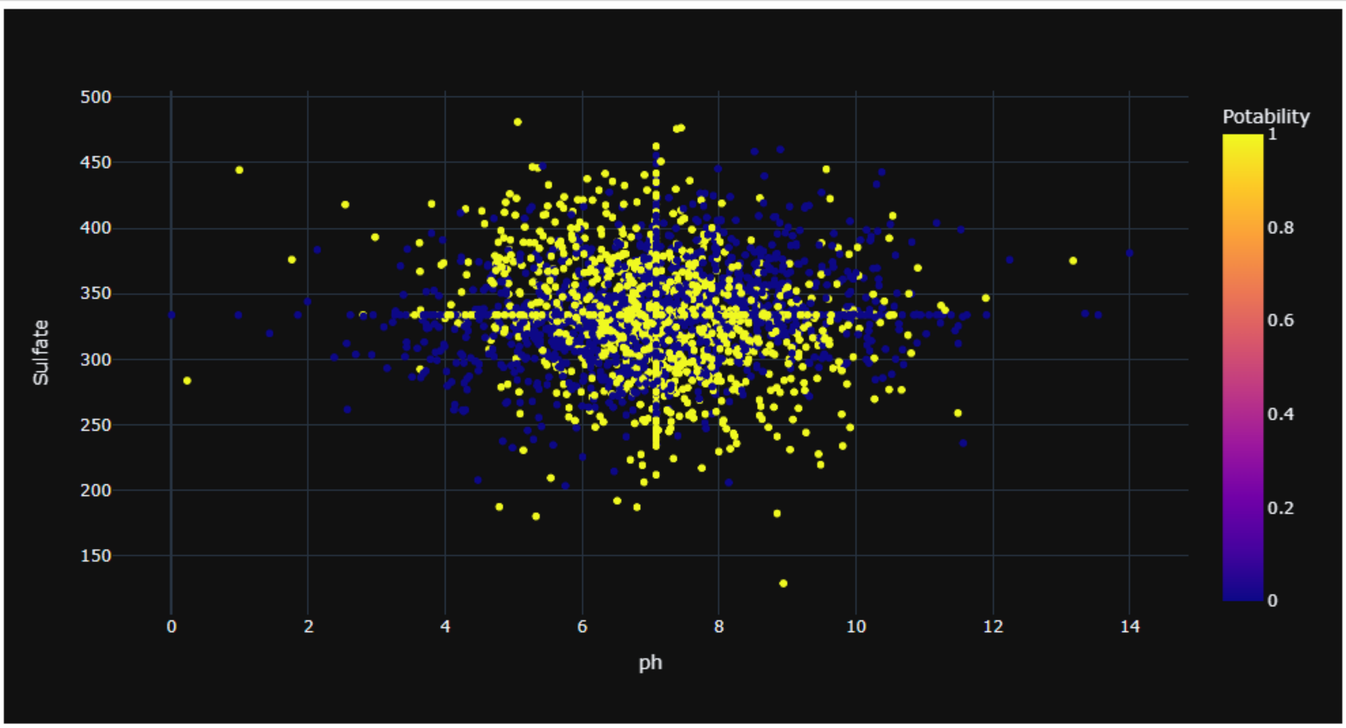

```
[197]: fig=px.pie(df,names='Potability',hole=0.1)
       fig.show()
```

5 plotly can draw scatter plots with any attributes

```
[198]: fig=px.scatter(df,x='ph',y='Sulfate',color='Potability')
       fig.show()
```



0
1
39%
61%



Sulfate
ph
Potability
500
450
400
350
300
250
200
150
0
2
4
6
8
10
12
14
1
0.8
0.6
0.4
0.2
0

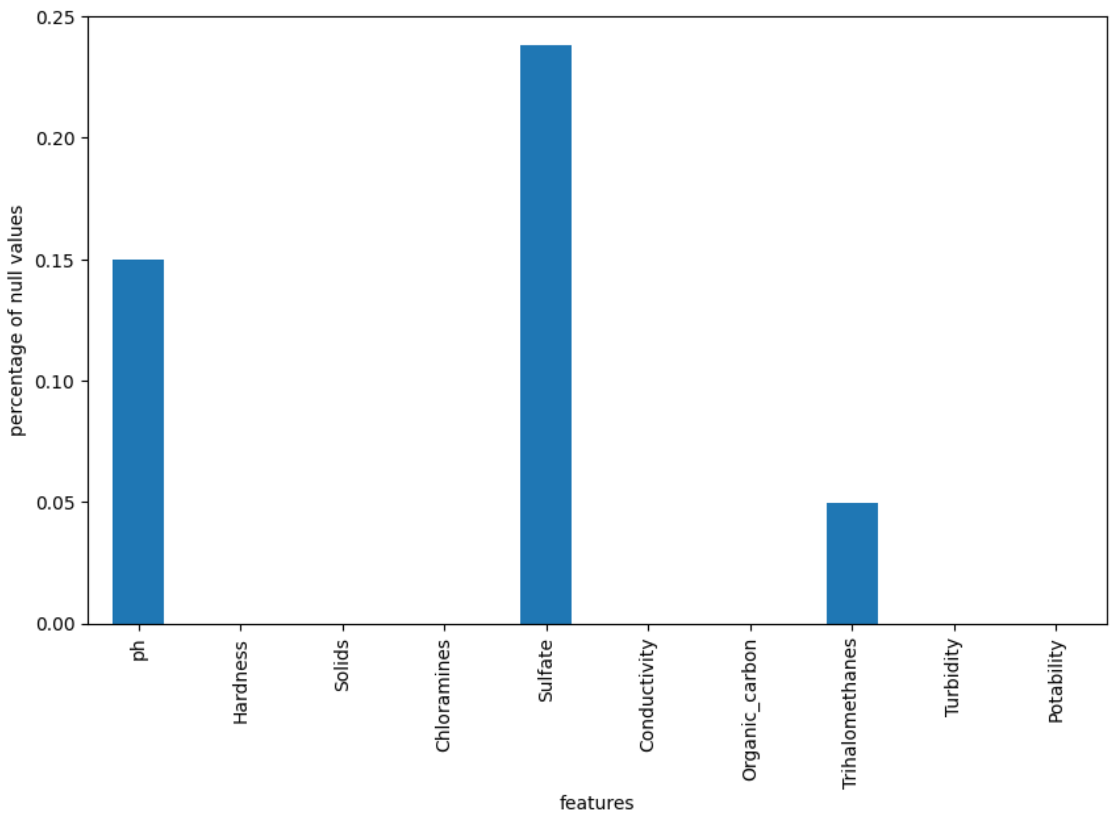
6 —visualization ends here—

7

8 managing the null values

[59]:
```
df.isnull().mean().plot.bar(figsize=(10,6))
plt.xlabel('features')
plt.ylabel('percentage of null values')
```

[59]: Text(0, 0.5, 'percentage of null values')



9 replacing the null values with their mean values

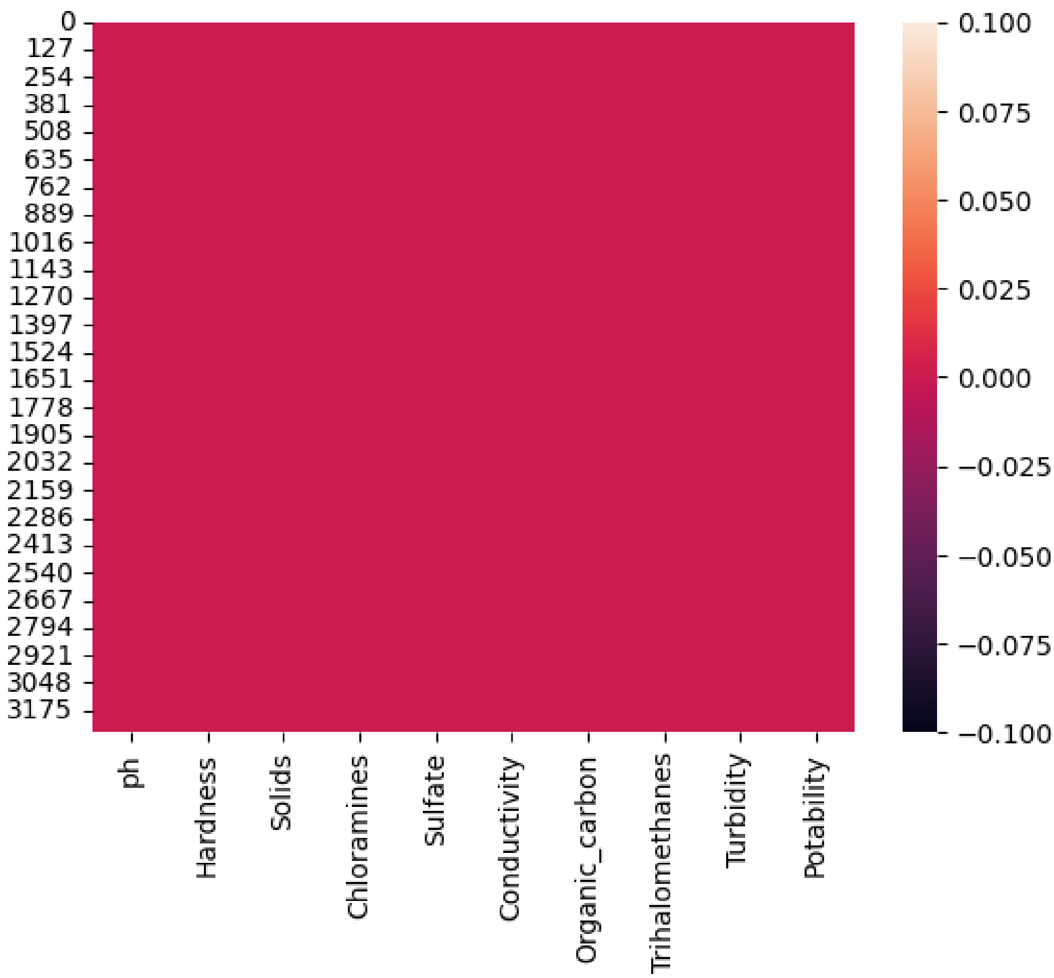
[60]:
```
df['ph']=df['ph'].fillna(df['ph'].mean())
df['Sulfate']=df['Sulfate'].fillna(df['Sulfate'].mean())
df['Trihalomethanes']=df['Trihalomethanes'].fillna(df['Trihalomethanes'].mean())
```

[61]:
```
df.isnull().sum()
```

[61]:
```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
dtype: int64
```

[62]:
```
sns.heatmap(df.isnull())
```

[62]: <Axes: >



10 features and target splitting , Scaling

[63]:
```
x=df.drop('Potability',axis=1)
y=df['Potability']
```

[64]:
```
x.shape,y.shape
```

[64]:
```
((3276, 9), (3276,))
```

[65]:
```
scaler=StandardScaler()
x=scaler.fit_transform(x)
```

[66]:
```
x
```

[66]:
```
array([[-6.04313345e-16,  2.59194711e-01, -1.39470871e-01, …,
        -1.18065057e+00,  1.30614943e+00, -1.28629758e+00],
       [-2.28933938e+00, -2.03641367e+00, -3.85986650e-01, …,
         2.70597240e-01, -6.38479983e-01,  6.84217891e-01],
       [ 6.92867789e-01,  8.47664833e-01, -2.40047337e-01, …,
         7.81116857e-01,  1.50940884e-03, -1.16736546e+00],
       …,
       [ 1.59125368e+00, -6.26829230e-01,  1.27080989e+00, …,
        -9.81329234e-01,  2.18748247e-01, -8.56006782e-01],
       [-1.32951593e+00,  1.04135450e+00, -1.14405809e+00, …,
        -9.42063817e-01,  7.03468419e-01,  9.50797383e-01],
       [ 5.40150905e-01, -3.85462310e-02, -5.25811937e-01, …,
         5.60940070e-01,  7.80223466e-01, -2.12445866e+00]])
```

11 train test split

[67]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

[68]:
```
x_train.shape,x_test.shape
```

[68]:
```
((2620, 9), (656, 9))
```

12 model building

13 1. logistic regression

[69]:
```
from sklearn.linear_model import LogisticRegression
#creating object
model_lr=LogisticRegression()
```

[70]:
```
#training
model_lr.fit(x_train,y_train)
```

[70]: LogisticRegression()

[71]:
```python
# predictions
pred_lr=model_lr.predict(x_test)
```

[72]:
```python
pred_lr
```

[72]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int64)
```

[73]:
```python
#accuracy score
accuracy_score_lr=accuracy_score(y_test,pred_lr)
accuracy_score_lr
```

[73]: 0.5929878048780488

14 2. Decision Tree

[74]:
```
from sklearn.tree import DecisionTreeClassifier
#object creation
model_dt=DecisionTreeClassifier(max_depth=4)
```

[75]:
```
#training
model_dt.fit(x_train,y_train)
```

[75]:
```
DecisionTreeClassifier(max_depth=4)
```

[76]:
```
#prediction
pred_dt=model_dt.predict(x_test)
```

[77]:
```
#accuracy
accuracy_score_dt=accuracy_score(y_test,pred_dt)
accuracy_score_dt
```

[77]:
```
0.6265243902439024
```

[78]:
```
#confusion matrix
cm2=confusion_matrix(y_test,pred_dt)
cm2 #367 are 0s predicted correctly,38 wrong !! 205 are 1s predicted␣
 ↪correctly,35 wrong
```

[78]:
```
array([[373,  15],
       [230,  38]], dtype=int64)
```

[79]:
```
# heatmap can be used to visualize
# sns.heatmap(cm2/np.sum(cm2))
```

15 3. Random Forest

[80]:
```
from sklearn.ensemble import RandomForestClassifier
model_rf=RandomForestClassifier()
```

[81]:
```
#training
model_rf.fit(x_train,y_train)
```

[81]:
```
RandomForestClassifier()
```

[82]:
```
#prediction
pred_rf=model_rf.predict(x_test)
```

[105]:
```
#accuracy
accuracy_score_rf=accuracy_score(y_test,pred_rf)
accuracy_score_rf*100
```

[105]: 68.59756097560977

[84]:
```
#confusion matrix
cm3=confusion_matrix(y_test,pred_rf)
cm3
```

[84]:
```
array([[357,  31],
       [175,  93]], dtype=int64)
```

16 4. k nearest neighbors (slowest)

[85]:
```
from sklearn.neighbors import KNeighborsClassifier

#creating model
# by default - model_knn=KNeighborsClassifier()
```

[86]:
```
# hyperparameter tuning
for i in range (4,15):
    model_knn=KNeighborsClassifier(n_neighbors=i)
    model_knn.fit(x_train,y_train)
    pred_knn=model_knn.predict(x_test)
    accuracy_score_knn=accuracy_score(y_test,pred_knn)
    print(i,accuracy_score_knn)
```

```
4 0.6402439024390244
5 0.6173780487804879
6 0.6265243902439024
7 0.6128048780487805
8 0.6189024390243902
9 0.6158536585365854
10 0.6265243902439024
11 0.6280487804878049
12 0.6280487804878049
13 0.6341463414634146
14 0.6341463414634146
```

17

[87]:
```
#accuracy
model_knn=KNeighborsClassifier(n_neighbors=10)
model_knn.fit(x_train,y_train)
pred_knn=model_knn.predict(x_test)
accuracy_score_knn=accuracy_score(y_test,pred_knn)
print(accuracy_score_knn)
```

```
0.6265243902439024
```

18 5. support vector machine

[88]:
```
from sklearn.svm import SVC#support vector classifier
```

[101]:
```
model_svm=SVC(kernel='rbf') #kernel types=linear,rbf(better accuracy),polynomial
```

[102]:
```
#training
model_svm.fit(x_train,y_train)
```

[102]:
```
SVC()
```

[103]:
```
# prediction
pred_svm=model_svm.predict(x_test)
```

[104]:
```
#accuracy
accuracy_score_svm=accuracy_score(y_test,pred_svm)
accuracy_score_svm*100
```

[104]:
```
67.53048780487805
```

[ ]:
```
# for kernel rbf-67.53048780487805,polynomial-61.4329268292683,for linear - 59.
↪14634146341463
```

19 6. AdaBoostClassifiers

[106]:
```
from sklearn.ensemble import AdaBoostClassifier
```

[156]:
```
#object creation
model_ada=AdaBoostClassifier(learning_rate=0.5,n_estimators=200)#hyperparameter␣
↪tunning-decrease learning rate for better accuracy,its by default 1
```

[157]:
```
#model training
model_ada.fit(x_train,y_train)
```

[157]:
```
AdaBoostClassifier(learning_rate=0.5, n_estimators=200)
```

[158]:
```
#prediction
pred_ada=model_ada.predict(x_test)
```

[159]:
```
#accuracy
accuracy_score_ada=accuracy_score(y_test,pred_ada)
accuracy_score_ada
```

[159]:
```
0.6189024390243902
```

[142]:
```
# by default accuracy is - 0.6097560975609756,after  hyperparameter tuning -␣
↪dose not effect much for this dataset
```

20 7. XGBoost

[146]:
```
from xgboost import XGBClassifier
```

[188]:
```
#object creation
model_xgb=XGBClassifier(n_estimators=200,learning_rate=0.4)
```

[189]:
```
#training
model_xgb.fit(x_train,y_train)
```

[189]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.4, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=200, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, …)
```

[190]:
```
#prediction
pred_xgb=model_xgb.predict(x_test)
```

[191]:
```
#accuracy
accuracy_score_xgb=accuracy_score(y_test,pred_xgb)
accuracy_score_xgb*100
```

[191]:
```
67.07317073170732
```

21 choosing the best model

[192]:
```
models=pd.DataFrame({
    'Model':['Logistic Regression','Decision Tree','Random␣
 ↪Forest','KNN','SVM','AdaBoost','XGBoost']
    , 'Accuracy Score' :␣
 ↪[accuracy_score_lr,accuracy_score_dt,accuracy_score_rf,accuracy_score_knn,accuracy_score_sv
})
```

[193]:
```
models
```

[193]:

	Model	Accuracy Score
0	Logistic Regression	0.592988
1	Decision Tree	0.626524
2	Random Forest	0.685976
3	KNN	0.626524

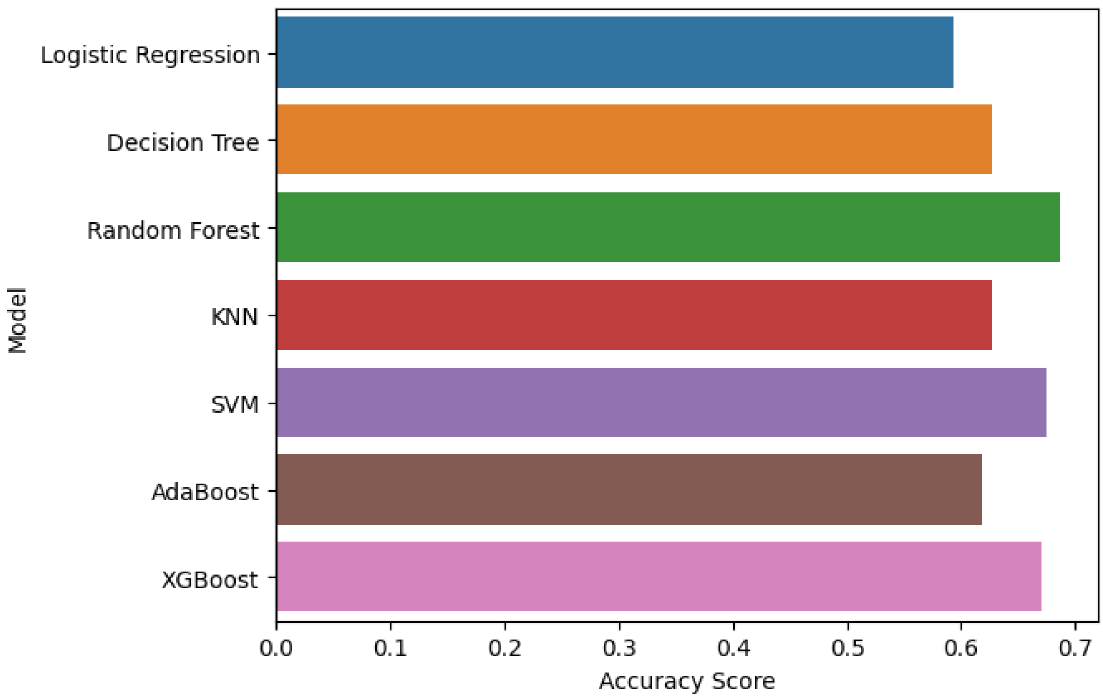
```
4                  SVM        0.675305
5             AdaBoost        0.618902
6              XGBoost        0.670732
```

[196]:

```
sns.barplot(x='Accuracy Score',y='Model',data=models)
models.sort_values(by='Accuracy Score',ascending=False)
```

[196]:

	Model	Accuracy Score
2	Random Forest	0.685976
4	SVM	0.675305
6	XGBoost	0.670732
1	Decision Tree	0.626524
3	KNN	0.626524
5	AdaBoost	0.618902
0	Logistic Regression	0.592988



[ ]:

```
#Random Forest and SVM are givving the highest accuracy
```