```
[ ]
# Import all the required libraries which are used to train the model or visualise the data
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import GridSearchCV  #hyper parameter tuning

#  Read the data
df=pd.read_csv('/content/water_potability.csv')
df
```

	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
...	...	...	...	...	...	...	...	...	...	...
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	NaN	392.449580	19.903225	NaN	2.798243	1
3273	9.419510	175.762646	33155.578218	7.350233	NaN	432.044783	11.039070	69.845400	3.298875	1
3274	5.126763	230.603758	11983.869376	6.303357	NaN	402.883113	11.168946	77.488213	4.708658	1
3275	7.874671	195.102299	17404.177061	7.509306	NaN	327.459760	16.140368	78.698446	2.309149	1

3276 rows × 10 columns

```
[ ]  1  # to display first 5 rows
     2  df.head()
```

	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

```
[ ]  1  # to display last 5 rows
     2  df.tail()
```

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
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	NaN	392.449580	19.903225	NaN	2.798243	1
3273	9.419510	175.762646	33155.578218	7.350233	NaN	432.044783	11.039070	69.845400	3.298875	1
3274	5.126763	230.603758	11983.869376	6.303357	NaN	402.883113	11.168946	77.488213	4.708658	1
3275	7.874671	195.102299	17404.177061	7.509306	NaN	327.459760	16.140368	78.698446	2.309149	1

```
[ ]  1  # to print the size
     2  print(df.shape)
```

```
(3276, 10)
```

```
[ ]  1  # find the column names
     2  print(df.columns)
```

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

```
[ ]  1  #  Describe the dataset which shows the minimum value, maximum value, mean value, count,standard deviation, etc.
     2  df.describe()
     3
```

	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

```
[ ]  1  # to find the datatypes
     2  df.dtypes
```

```
ph                float64
Hardness          float64
Solids            float64
Chloramines       float64
Sulfate           float64
Conductivity      float64
Organic_carbon    float64
Trihalomethanes   float64
Turbidity         float64
Potability          int64
dtype: object
```

```
[ ]  1  # Checking for missing values
     2  df.isna().sum()
```

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

```
[ ]  1  #  Filling the missing values using a mean value of each feature
     2  df['ph']=df['ph'].fillna(df['ph'].mean())
     3  df['Sulfate']=df['Sulfate'].fillna(df['Sulfate'].mean())
     4  df['Trihalomethanes']=df['Trihalomethanes'].fillna(df['Trihalomethanes'].mean())
     5  df.isna().sum()
```

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

```
[ ]  1  #Checking the value counts of our target feature Potability.
     2  df.Potability.value_counts()
```

```
0    1998
1    1278
Name: Potability, dtype: int64
```

```
[ ]  1  #Checking the value counts of our target feature Potability.
     2  df.Potability.value_counts()
```

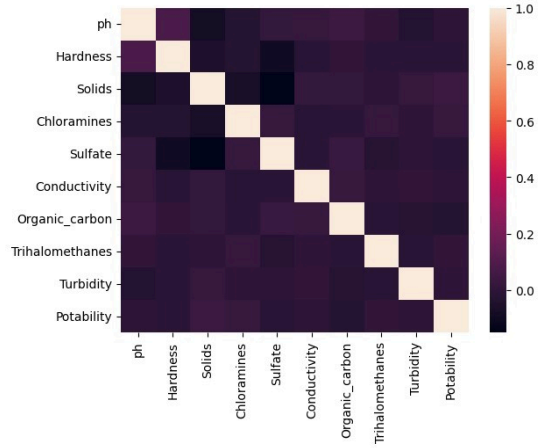
```
0    1998
1    1278
Name: Potability, dtype: int64
```

```
1  df.corr() #find the correltion(how to relate one column related to another column)
```

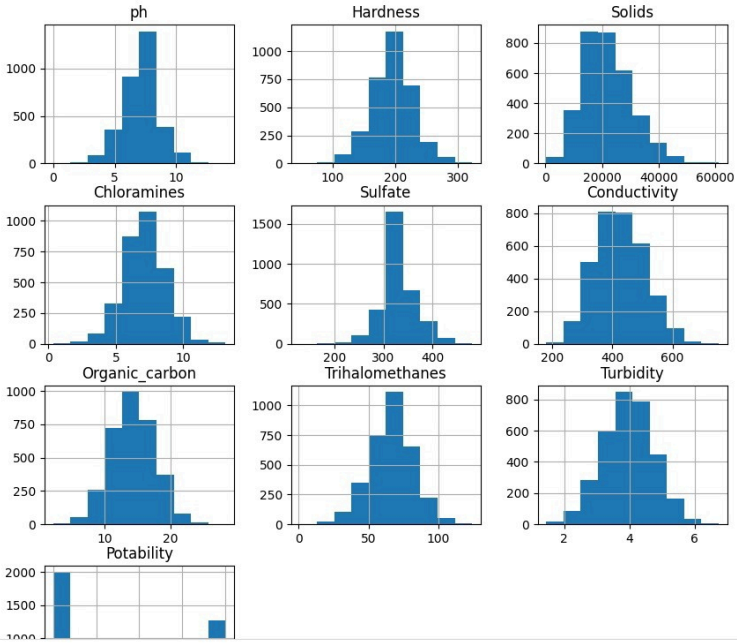
	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turbidity	Potability
ph	1.000000	0.075833	-0.081884	-0.031811	0.014403	0.017192	0.040061	0.002994	-0.036222	-0.003287
Hardness	0.075833	1.000000	-0.046899	-0.030054	-0.092766	-0.023915	0.003610	-0.012690	-0.014449	-0.013837
Solids	-0.081884	-0.046899	1.000000	-0.070148	-0.149840	0.013831	0.010242	-0.008875	0.019546	0.033743
Chloramines	-0.031811	-0.030054	-0.070148	1.000000	0.023791	-0.020486	-0.012653	0.016627	0.002363	0.023779
Sulfate	0.014403	-0.092766	-0.149840	0.023791	1.000000	-0.014059	0.026909	-0.025605	-0.009790	-0.020619
Conductivity	0.017192	-0.023915	0.013831	-0.020486	-0.014059	1.000000	0.020966	0.001255	0.005798	-0.008128
Organic_carbon	0.040061	0.003610	0.010242	-0.012653	0.026909	0.020966	1.000000	-0.012976	-0.027308	-0.030001
Trihalomethanes	0.002994	-0.012690	-0.008875	0.016627	-0.025605	0.001255	-0.012976	1.000000	-0.021502	0.006960
Turbidity	-0.036222	-0.014449	0.019546	0.002363	-0.009790	0.005798	-0.027308	-0.021502	1.000000	0.001581
Potability	-0.003287	-0.013837	0.033743	0.023779	-0.020619	-0.008128	-0.030001	0.006960	0.001581	1.000000

```
1
2 sns.heatmap(df.corr()) #graw a graph (heatmap) of corelation
```

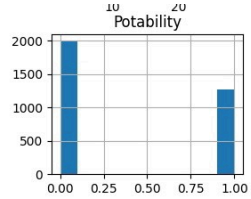
```
<Axes: >
```



```
plt.rcParams['figure.figsize'] = [10,10]
df.hist()
plt.show()
# defines a runtime configuration (rc) containing the default styles for every plot element you create
```

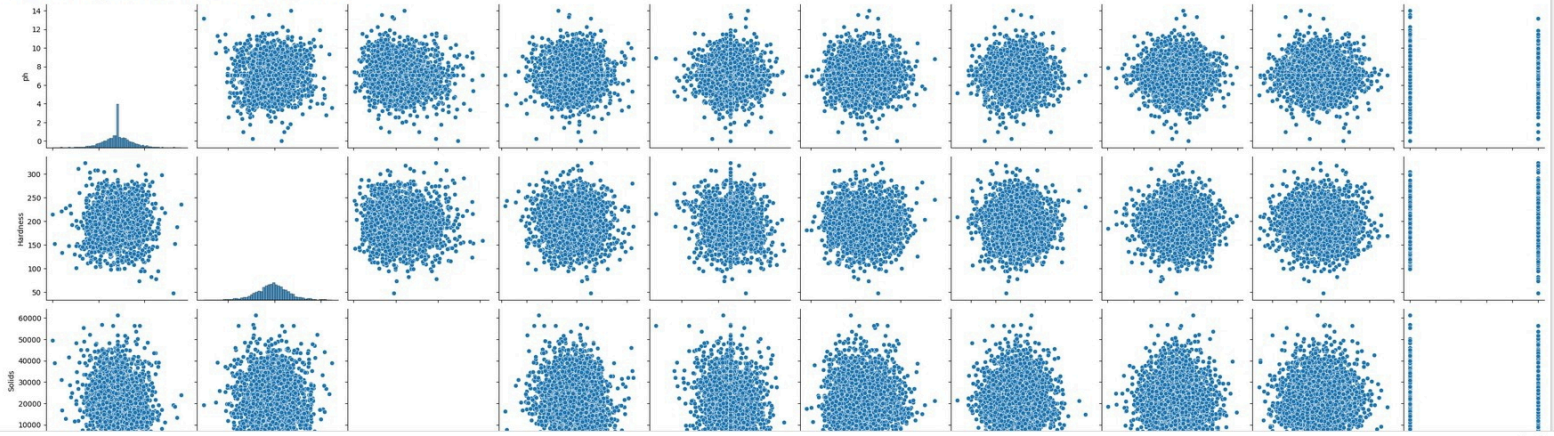


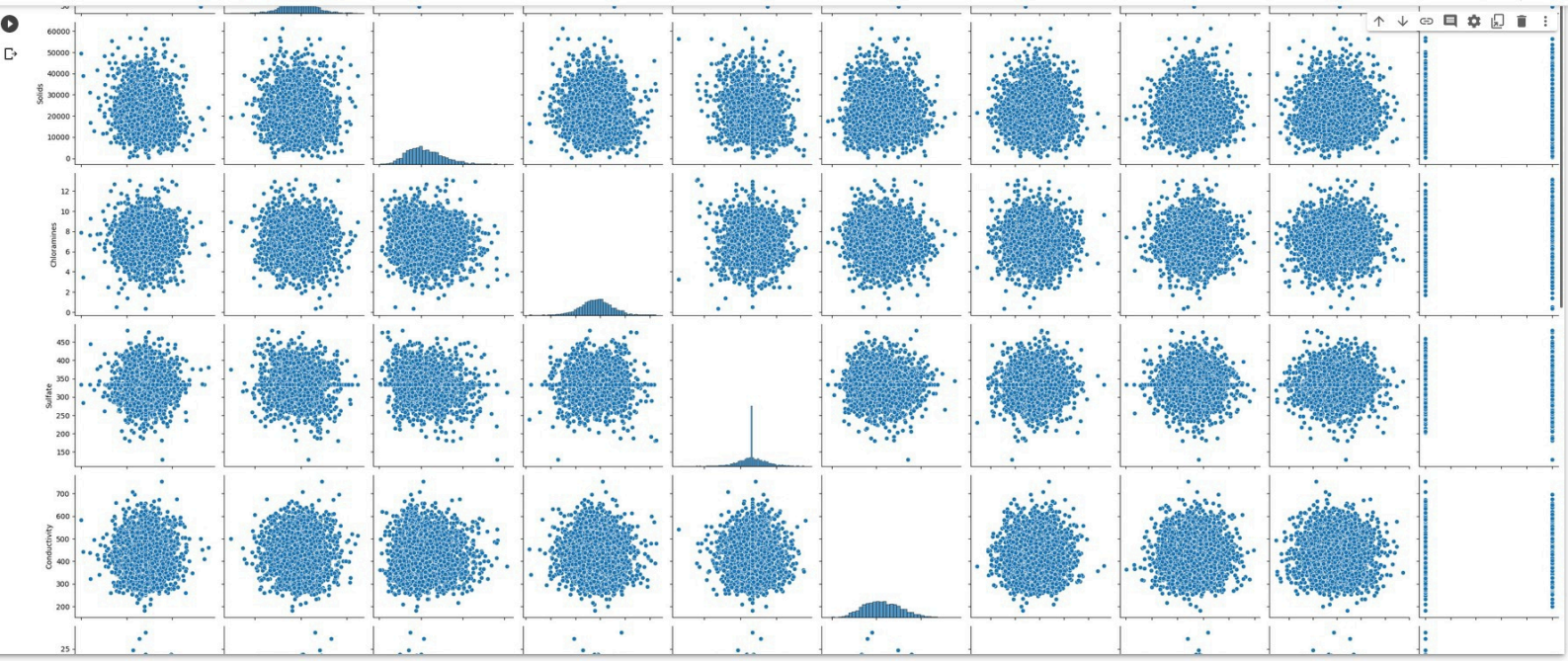
[ ]

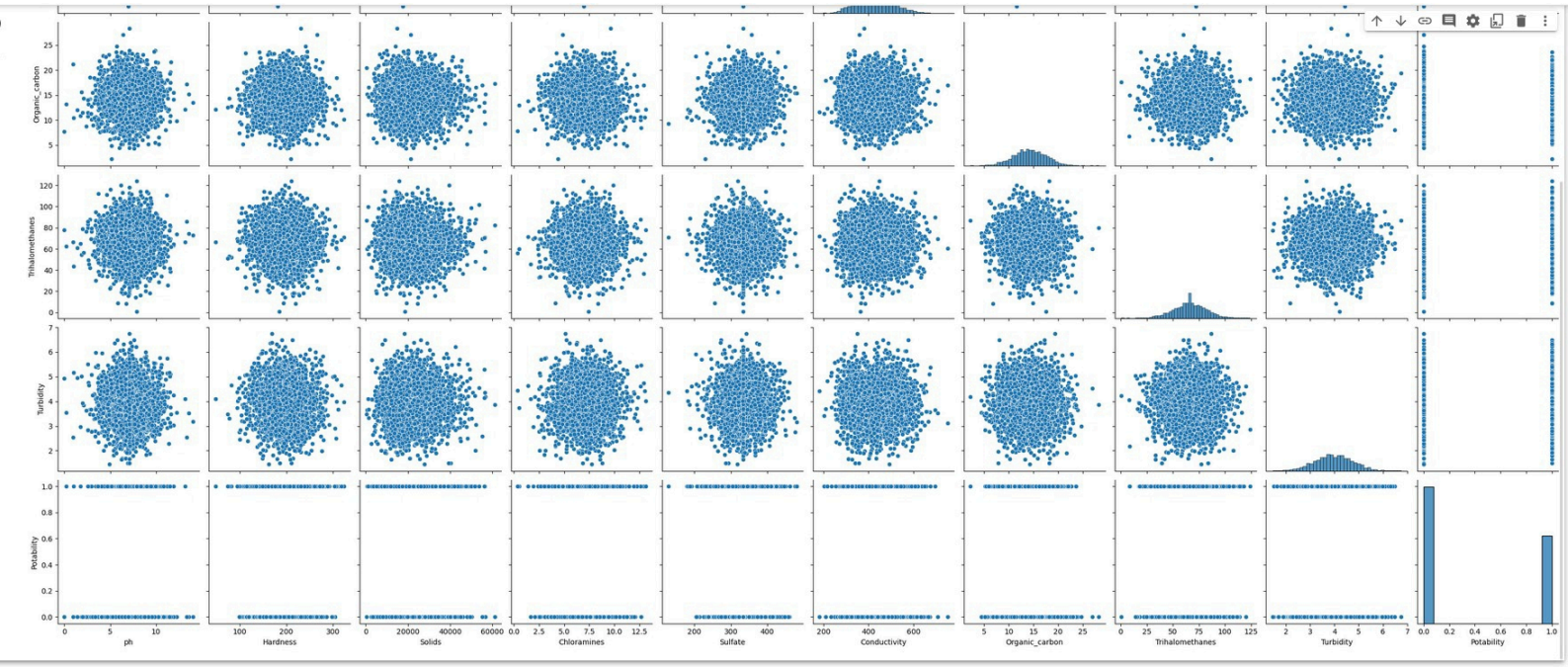



```
1  sns.pairplot(df,height=5,size=3)
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/axisgrid.py:2095: UserWarning: The `size` parameter has been renamed to `height`; please update your code.
  warnings.warn(msg, UserWarning)
<seaborn.axisgrid.PairGrid at 0x7fdf015544c0>
```

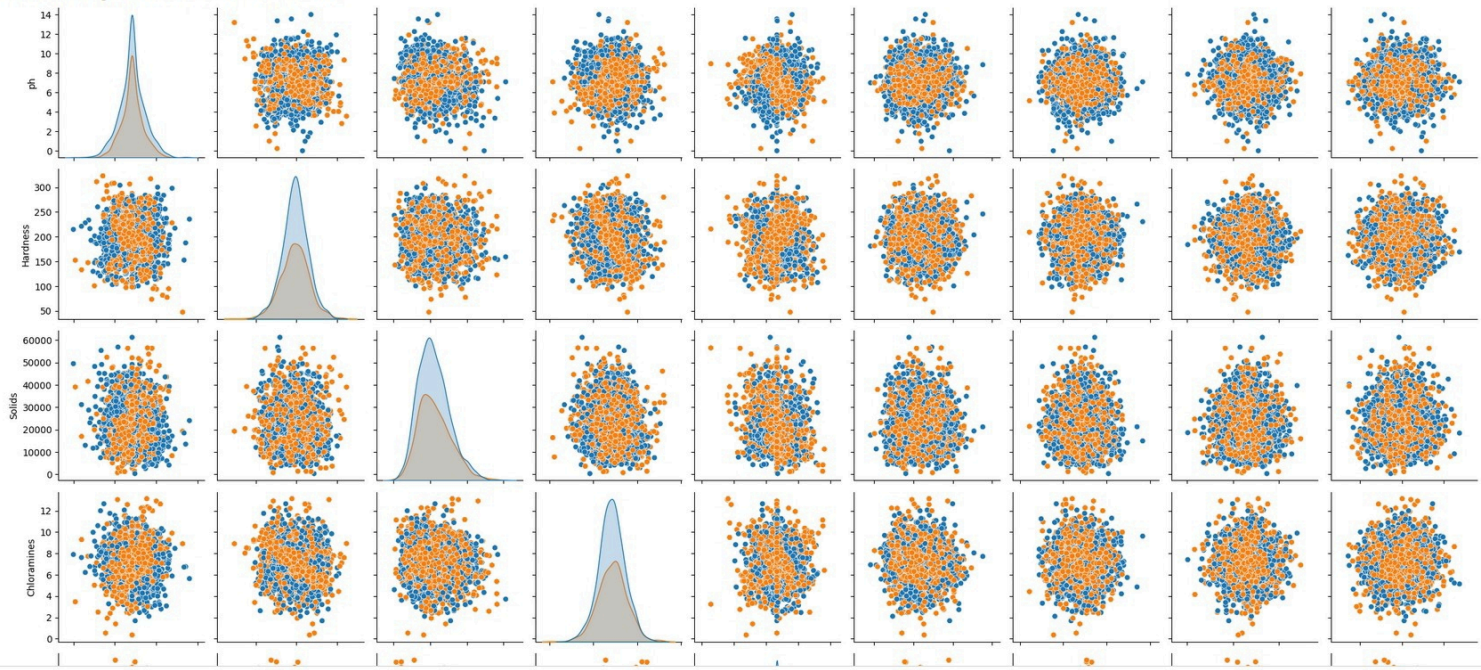



Solids
Chloramines
Sulfate
Conductivity

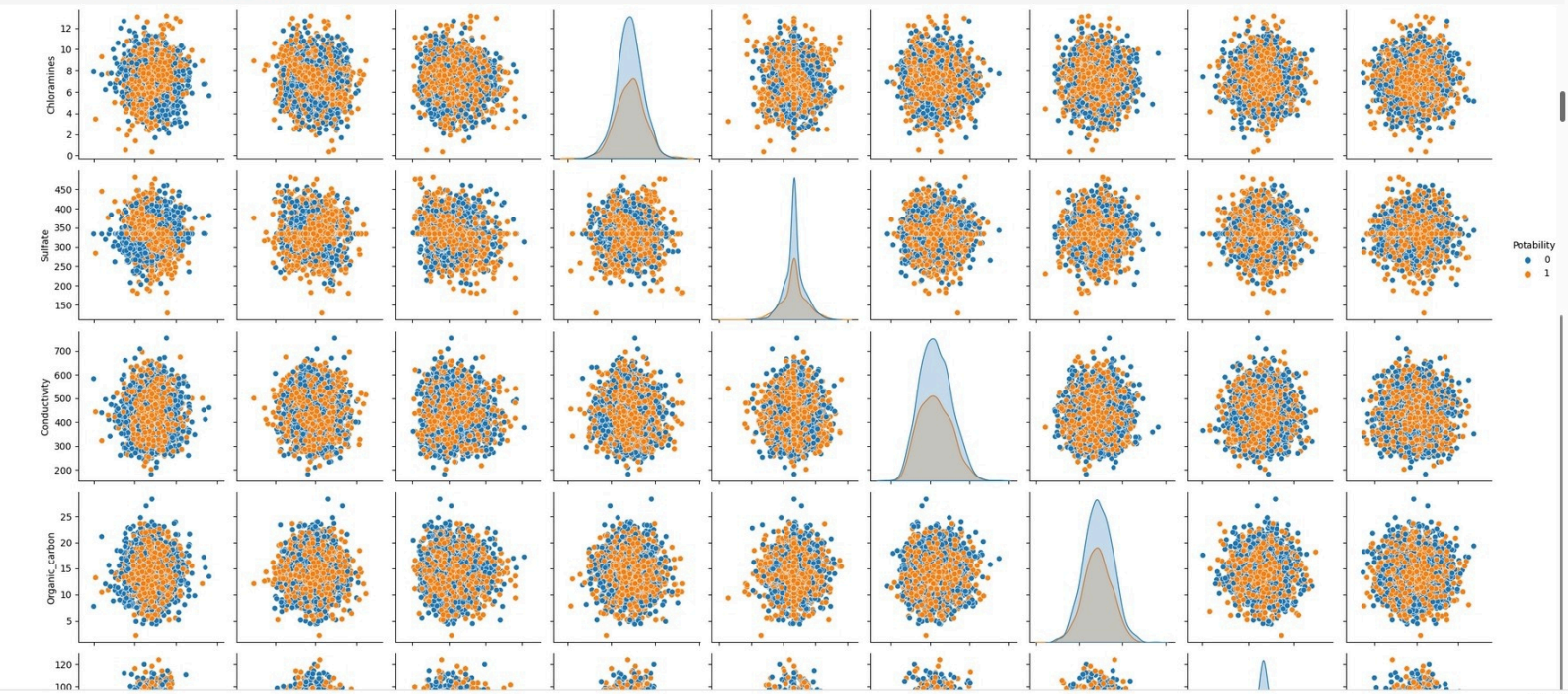

Organic_carbon
Trihalomethanes
Turbidity
Potability
ph
Hardness
Solids
Chloramines
Sulfate
Conductivity
Organic_carbon
Trihalomethanes
Turbidity
Potability

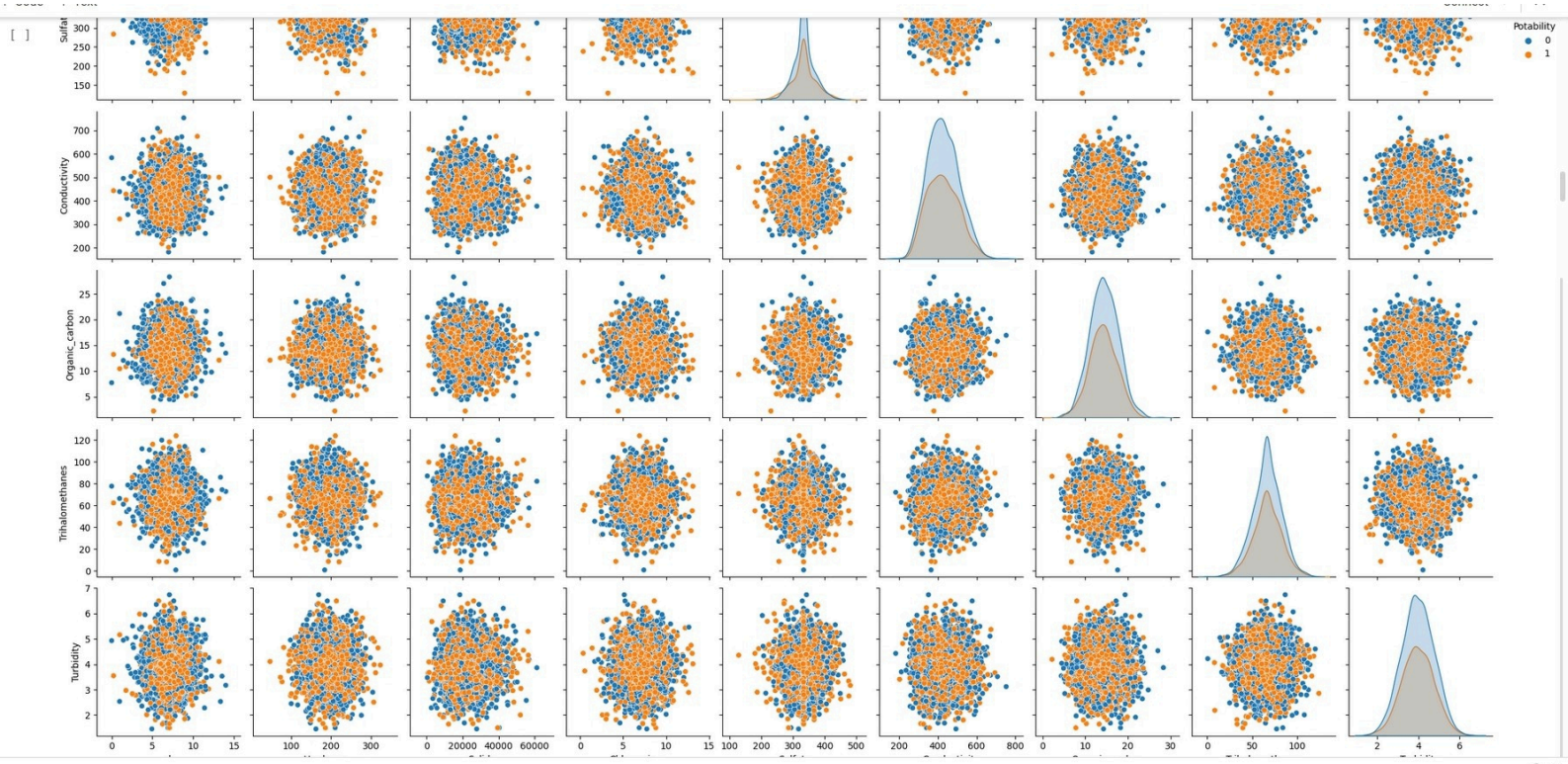
```
[ ]  1  sns.pairplot(df,hue="Potability")
```

```
<seaborn.axisgrid.PairGrid at 0x7fdf0e9c1210>
```

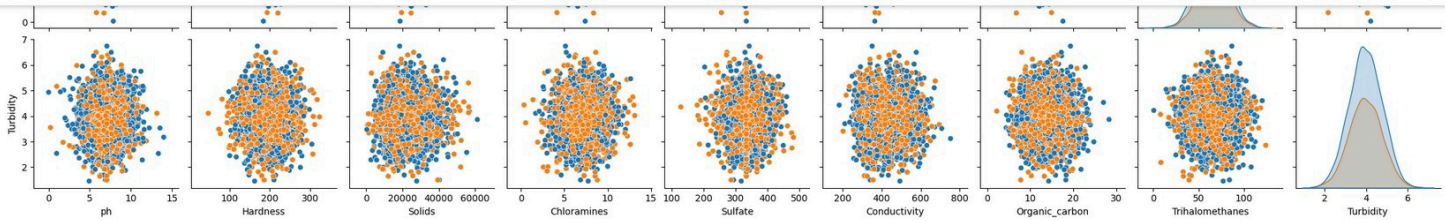


```
] 1 sns.pairplot(df,hue="Potability")
```



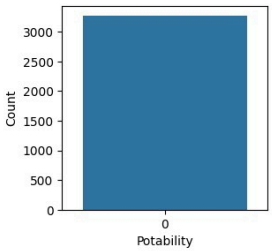

Potability
0
1
Sulfate
Conductivity
Organic_carbon
Trihalomethanes
Turbidity

[ ]



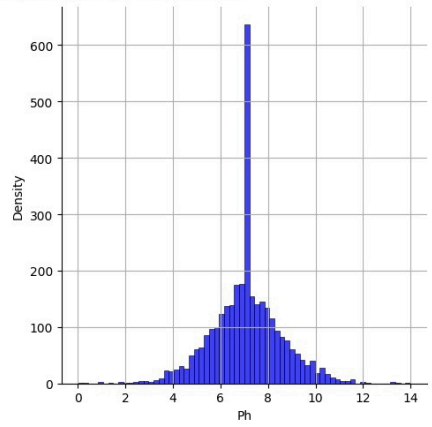
[ ]

```
# Visualising the potability using a countplot function.
plt.figure(figsize=(3,3))
sns.countplot(df['Potability'])
plt.xlabel('Potability',fontsize=10)
plt.ylabel('Count',fontsize=10)
plt.show()
```



```
[ ]  1  #  visualising the pH value using a displot function.
     2  plt.figure(figsize=(3,3))
     3  sns.displot(df['ph'],color='blue')
     4  plt.xlabel('Ph',fontsize=10)
     5  plt.ylabel('Density',fontsize=10)
     6  plt.grid()
     7  plt.show()
     8
```

```
<Figure size 300x300 with 0 Axes>
```



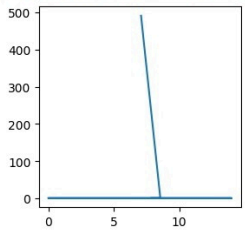
Find the vaLue count and plot

```
ph=df['ph'].value_counts()
ph
```

```
7.080795    491
8.554097      1
6.538084      1
5.915807      1
8.136498      1
           ...
4.187491      1
7.808012      1
5.895949      1
7.269652      1
7.874671      1
Name: ph, Length: 2786, dtype: int64
```

```
plt.figure(figsize=(3,3))
plt.plot(ph)
```

```
[<matplotlib.lines.Line2D at 0x7fdef853ce50>]
```



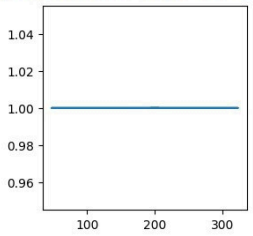
+ Code + Text Connect

```
[ ]  1  Hardness=df['Hardness'].value_counts()
     2  Hardness
```

```
204.890455    1
134.560276    1
170.190912    1
237.461099    1
171.238926    1
             ..
218.237186    1
208.374188    1
142.145566    1
179.799917    1
195.102299    1
Name: Hardness, Length: 3276, dtype: int64
```

```
[ ]  1  plt.figure(figsize=(3,3))
     2  plt.plot(Hardness)
```

```
[<matplotlib.lines.Line2D at 0x7fdef8586d70>]
```

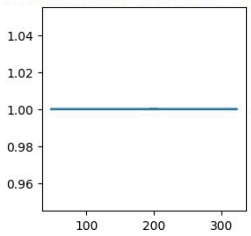


```
[ ]  1  Hardness=df['Hardness'].value_counts()
     2  Hardness
```

```
204.890455    1
134.560276    1
170.190912    1
237.461099    1
171.238926    1
             ..
218.237186    1
208.374188    1
142.145566    1
179.799917    1
195.102299    1
Name: Hardness, Length: 3276, dtype: int64
```

```
[ ]  1  plt.figure(figsize=(3,3))
     2  plt.plot(Hardness)
```

```
[<matplotlib.lines.Line2D at 0x7fdef8586d70>]
```

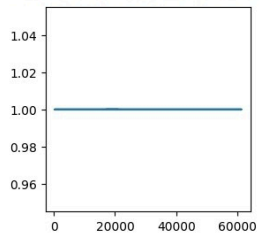


```
Solids=df['Solids'].value_counts()
Solids
```

```
20791.318981    1
15979.334793    1
37000.955674    1
18736.190902    1
12289.900922    1
               ..
22824.699465    1
21809.709834    1
45141.686036    1
17037.725367    1
17404.177061    1
Name: Solids, Length: 3276, dtype: int64
```

```
plt.figure(figsize=(3,3))
plt.plot(Solids)
```

```
[<matplotlib.lines.Line2D at 0x7fdef85e7ee0>]
```



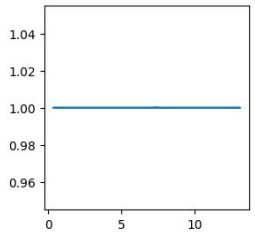
+ Code + Text Connect

```
Chloramines=df['Chloramines'].value_counts()
Chloramines
```

```
7.300212     1
9.504361     1
6.217223     1
5.599870     1
10.786500    1
            ..
8.696479     1
5.846112     1
6.030640     1
6.378364     1
7.509306     1
Name: Chloramines, Length: 3276, dtype: int64
```

```
plt.figure(figsize=(3,3))
plt.plot(Chloramines)
```

```
[<matplotlib.lines.Line2D at 0x7fdef8465f90>]
```

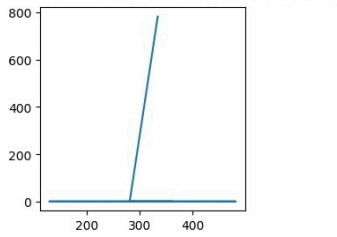


```
1  Sulfate=df['Sulfate'].value_counts()
2  Sulfate
```

```
333.775777    781
280.745623      1
332.744519      1
391.918229      1
330.905370      1
              ...
343.620823      1
359.710517      1
389.219586      1
337.231469      1
359.948574      1
Name: Sulfate, Length: 2496, dtype: int64
```

```
[ ]  1  plt.figure(figsize=(3,3))
     2  plt.plot(Sulfate)
```

```
[<matplotlib.lines.Line2D at 0x7fdef84bb130>]
```

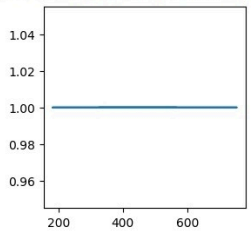


```
[ ]  1
     2  Conductivity=df['Conductivity'].value_counts()
     3  Conductivity
```

```
564.308654    1
418.642063    1
517.576762    1
235.042283    1
501.559725    1
             ..
521.016261    1
264.508083    1
369.280429    1
590.060546    1
327.459760    1
Name: Conductivity, Length: 3276, dtype: int64
```

```
[ ]  1  plt.figure(figsize=(3,3))
     2  plt.plot(Conductivity)
```

```
[<matplotlib.lines.Line2D at 0x7fdef83389d0>]
```

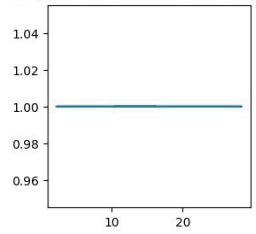


```
[ ]  1  Organic_carbon=df['Organic_carbon'].value_counts()
     2  Organic_carbon
```

```
10.379783    1
12.897635    1
15.871770    1
11.545477    1
12.284334    1
            ..
15.310631    1
11.235144    1
20.605552    1
12.092499    1
16.140368    1
Name: Organic_carbon, Length: 3276, dtype: int64
```

```
[ ]  1  plt.figure(figsize=(3,3))
     2  plt.plot(Organic_carbon)
```

```
[<matplotlib.lines.Line2D at 0x7fdef8389d20>]
```

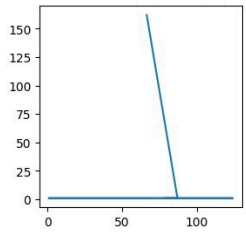


```
[ ]  1  Trihalomethanes=df['Trihalomethanes'].value_counts()
     2  Trihalomethanes
```

```
66.396293    162
86.990970      1
56.715510      1
77.730814      1
90.394895      1
             ...
73.723070      1
46.682597      1
70.168389      1
81.592362      1
78.698446      1
Name: Trihalomethanes, Length: 3115, dtype: int64
```

```
[ ]  1
     2  plt.figure(figsize=(3,3))
     3  plt.plot(Trihalomethanes)
```

```
[<matplotlib.lines.Line2D at 0x7fdef83e2e90>]
```

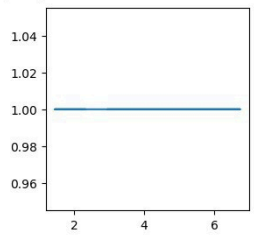


```
[ ]  1  Turbidity=df['Turbidity'].value_counts()
     2  Turbidity
```

```
2.963135    1
3.987012    1
4.066229    1
3.759326    1
4.876273    1
           ..
3.741143    1
4.592959    1
4.604725    1
4.165550    1
2.309149    1
Name: Turbidity, Length: 3276, dtype: int64
```

```
[ ]  1
     2  plt.figure(figsize=(3,3))
     3  plt.plot(Turbidity)
```

```
[<matplotlib.lines.Line2D at 0x7fdef826cd90>]
```



```
[ ]  1  # to print the datatypes
     2  df.dtypes
```

```
ph                 float64
Hardness           float64
Solids             float64
Chloramines        float64
Sulfate            float64
Conductivity       float64
Organic_carbon     float64
Trihalomethanes    float64
Turbidity          float64
Potability           int64
dtype: object
```

```
[ ]  1  # Assigning input and output
     2  x=df.iloc[:,:-1].values
     3  print(x)
     4  y=df.iloc[:,-1].values
     5  y
```

```
[[7.08079450e+00 2.04890455e+02 2.07913190e+04 ... 1.03797831e+01
  8.69909705e+01 2.96313538e+00]
 [3.71608008e+00 1.29422921e+02 1.86300579e+04 ... 1.51800131e+01
  5.63290763e+01 4.50065627e+00]
 [8.09912419e+00 2.24236259e+02 1.99095417e+04 ... 1.68686369e+01
  6.64200925e+01 3.05593375e+00]
 ...
 [9.41951032e+00 1.75762646e+02 3.31555782e+04 ... 1.10390697e+01
  6.98454003e+01 3.29887550e+00]
 [5.12676292e+00 2.30603758e+02 1.19838694e+04 ... 1.11689462e+01
  7.74882131e+01 4.70865847e+00]
 [7.87467136e+00 1.95102299e+02 1.74041771e+04 ... 1.61403676e+01
  7.86984463e+01 2.30914906e+00]]
array([0, 0, 0, ..., 1, 1, 1])
```

Modelling

Modelling

```
[ ]  1  from sklearn.model_selection import train_test_split
     2  x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=42)
```

Normalization

```
[ ]  1  from sklearn.preprocessing import StandardScaler
     2  scaler=StandardScaler()
     3  scaler.fit(x_train)
     4  x_train=scaler.transform(x_train)
     5  x_test=scaler.transform(x_test)
     6  x_train
```

```
array([[-0.0089824 , -0.24699831,  0.75053604, ..., -1.11745208,
        -0.20234542, -0.93893423],
       [ 0.07488172, -0.85645535,  0.08009625, ...,  0.65600216,
        -0.12232603,  3.12613141],
       [-0.0089824 ,  1.4101707 ,  0.80745866, ..., -2.68242954,
        -0.00327098, -0.56055613],
       ...,
       [-0.31698661, -0.29832594,  0.9918901 , ..., -0.13267988,
         1.27879668, -0.60170937],
       [ 1.15509923,  0.65460822, -0.66652007, ...,  0.63894108,
        -1.21848478,  0.40600947],
       [-0.27063182,  0.05287723,  1.41188532, ..., -0.63085449,
        -0.00327098, -0.25756054]])
```

Algorithms that are used in this project.

KNeighbours(hyper parameter tuning)

Naive bayes

SVM

Random Forest

Decision Tree

1. KNN Algorithm

```
[ ]
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train,y_train)
y_predict=knn.predict(x_test)
y_predict
print(knn.predict([[7.8,205.56,30700.55,5.2,400.56,66600.2,9.4,89.6,4.6]]))
```

```
[0]
```

```
[ ]
# hyper parameter tuning

cls1=KNeighborsClassifier()  #create object
params={'n_neighbors':[3,5,7,9,8,6,11,10],'weights':['uniform','distance']}
clf=GridSearchCV(cls1,params,cv=10,scoring='accuracy')  #cv:cross validtaion
clf.fit(x_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: KNeighborsClassifier
    ▸ KNeighborsClassifier
```

```
print(clf.best_params_)
```

```
{'n_neighbors': 8, 'weights': 'distance'}
```

```

cls2=KNeighborsClassifier(n_neighbors=9,weights='distance',algorithm = 'brute',metric = 'minkowski')
cls2.fit(x_train,y_train)
y_pred=cls2.predict(x_test)
y_pred
```

```
array([0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 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, 1, 1, 1, 1, 0, 1,
       1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
```

```
[ ]
```

```
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1,
0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1])
```

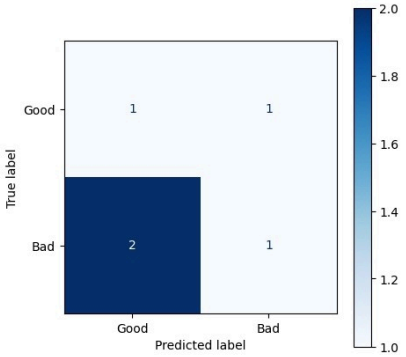
```
[ ]
```

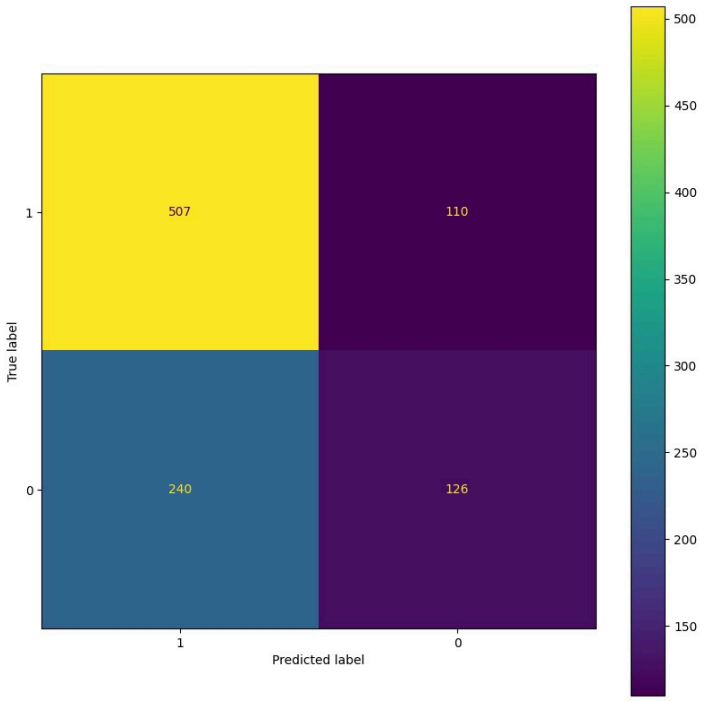
```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report,ConfusionMatrixDisplay
cnf=confusion_matrix(y_test,y_pred)
print(cnf)
score=accuracy_score(y_pred,y_test)
print("accuracy_score is",score)
report=classification_report(y_pred,y_test)
print(report)
```

```
[[507 110]
 [240 126]]
accuracy_score is 0.6439471007121058
              precision    recall  f1-score   support

           0       0.82      0.68      0.74       747
           1       0.34      0.53      0.42       236

    accuracy                           0.64       983
   macro avg       0.58      0.61      0.58       983
weighted avg       0.71      0.64      0.67       983
```

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
label=[1,0]
disp = ConfusionMatrixDisplay.from_predictions(
                [0,1,1,0,1],
                [0,1,0,1,0],
                cmap=plt.cm.Blues,
                display_labels=['Good','Bad'],
                values_format='',
)
fig = disp.ax_.get_figure()
fig.set_figwidth(5)
fig.set_figheight(5)
plt=ConfusionMatrixDisplay(cnf,display_labels=label)
plt.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fdef829a950>
```




507
110
240
126
1
0
True label
1
0
Predicted label
500
450
400
350
300
250
200
150

2. Naive Bayes Algorithm

```
[ ]  1  from sklearn.naive_bayes import GaussianNB
     2  model1=GaussianNB()
     3  model1.fit(x_train,y_train)
     4  y1_pred=model1.predict(x_test)
     5  print(model1.predict([[7.8,205.56,30700.55,5.2,400.56,66600.2,9.4,89.6,4.6]]))
```

```
[1]
```

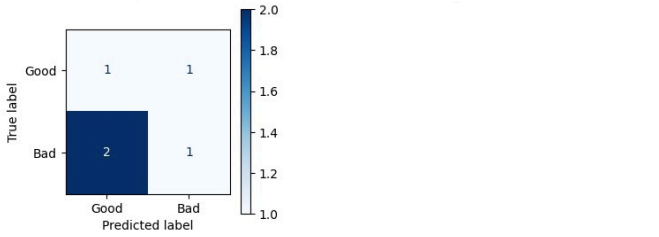
```
[ ]  1
     2  from sklearn.metrics import confusion_matrix,accuracy_score,classification_report,ConfusionMatrixDisplay
     3  cnf1=confusion_matrix(y_test,y1_pred)
     4  print(cnf1)
     5  score1=accuracy_score(y1_pred,y_test)
     6  print("accuracy_score is",score1)
     7  report1=classification_report(y1_pred,y_test)
     8  print(report1)
     9
```

```
[[546  71]
 [290  76]]
accuracy_score is 0.6327568667344863
              precision    recall  f1-score   support

           0       0.88      0.65      0.75       836
           1       0.21      0.52      0.30       147

    accuracy                           0.63       983
   macro avg       0.55      0.59      0.52       983
weighted avg       0.78      0.63      0.68       983
```

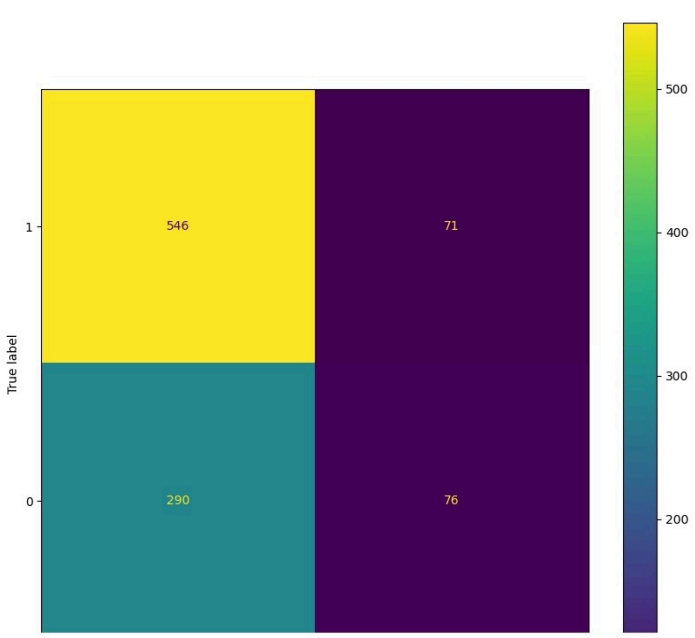
```
[ ]
```

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
label=[1,0]
disp = ConfusionMatrixDisplay.from_predictions(
                    [0,1,1,0,1],
                    [0,1,0,1,0],
                    cmap=plt.cm.Blues,
                    display_labels=['Good','Bad'],
                    values_format='',
)
fig = disp.ax_.get_figure()
fig.set_figwidth(3)
fig.set_figheight(3)
plt=ConfusionMatrixDisplay(cnf1,display_labels=label)
plt.plot()

```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fdef838afe0>
```



[ ]


546
71
290
76
1
0
True label
500
400
300
200

3. Support Vector Machine Algorithm

```
[ ]  1  from sklearn.svm import SVC
     2  model2=SVC()
     3  model2.fit(x_train,y_train)
     4  y2_pred=model2.predict(x_test)
     5  print(model2.predict([[7.8,205.56,30700.55,5.2,400.56,66600.2,9.4,89.6,4.6]]))
```

```
[1]
```

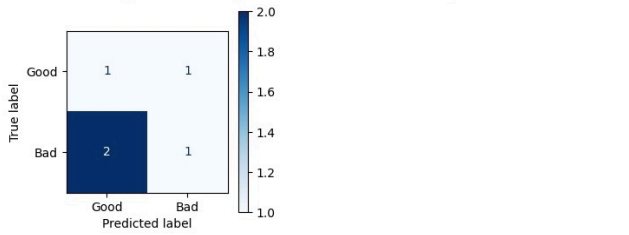
```
[ ]  1  from sklearn.metrics import confusion_matrix,accuracy_score
     2  cnf2=confusion_matrix(y_test,y2_pred)
     3  print(cnf2)
     4  score2=accuracy_score(y2_pred,y_test)
     5  print("accuracy_score is",score2)
     6  report2=classification_report(y2_pred,y_test)
     7  print(report2)
```

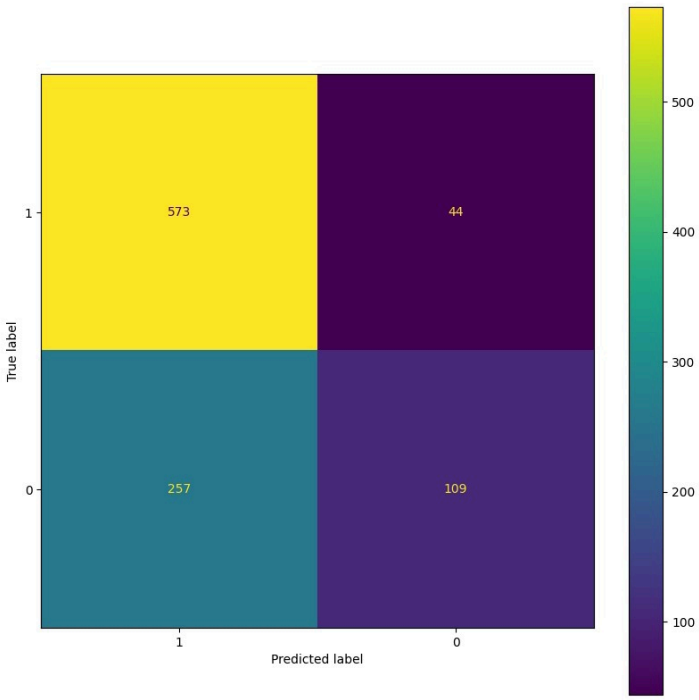
```
[[573  44]
 [257 109]]
accuracy_score is 0.6937945066124109
              precision    recall  f1-score   support

           0       0.93      0.69      0.79       830
           1       0.30      0.71      0.42       153

    accuracy                           0.69       983
   macro avg       0.61      0.70      0.61       983
weighted avg       0.83      0.69      0.73       983
```

```
[ ]
```

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
label=[1,0]
disp = ConfusionMatrixDisplay.from_predictions(
                    [0,1,1,0,1],
                    [0,1,0,1,0],
                    cmap=plt.cm.Blues,
                    display_labels=['Good','Bad'],
                    values_format='',
)
fig = disp.ax_.get_figure()
fig.set_figwidth(3)
fig.set_figheight(3)
plt=ConfusionMatrixDisplay(cnf2,display_labels=label)
plt.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fdef82442b0>
```




573
44
257
109
1
0
True label
1
0
Predicted label
500
400
300
200
100

4. Decision Tree Classifier

```
[ ]  1  from sklearn.tree import DecisionTreeClassifier
     2  model3 = DecisionTreeClassifier(max_depth=4, random_state=42)
     3  model3.fit(x_train,y_train)
     4  DecisionTreeClassifier()
     5  y3_pred = model3.predict(x_test)
     6  y3_pred
```

```
array([0, 0, 0, 0, 0, 1, 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, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 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, 1, 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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 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, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 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, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 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, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 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, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 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, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0,
```

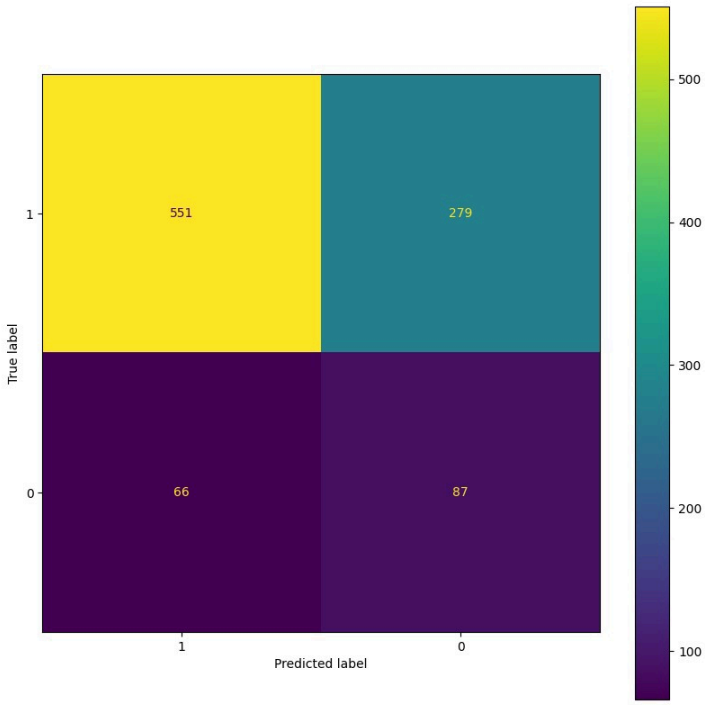
```
[ ]
from sklearn.metrics import confusion_matrix,accuracy_score,ConfusionMatrixDisplay
cnf3=confusion_matrix(y3_pred,y_test)
print(cnf3)
score3=accuracy_score(y3_pred,y_test)
print("accuracy_score is",score3)
report3=classification_report(y3_pred,y_test)
print(report3)
```

```
[[551 279]
 [ 66  87]]
accuracy_score is 0.6490335707019329
              precision    recall  f1-score   support

           0       0.89      0.66      0.76       830
           1       0.24      0.57      0.34       153

    accuracy                           0.65       983
   macro avg       0.57      0.62      0.55       983
weighted avg       0.79      0.65      0.70       983
```

```
[ ]
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
label=[1,0]
disp = ConfusionMatrixDisplay.from_predictions(
                    [0,1,1,0,1],
                    [0,1,0,1,0],
                    cmap=plt.cm.Blues,
                    display_labels=['Good','Bad'],
                    values_format='',
)
fig = disp.ax_.get_figure()
fig.set_figwidth(1)
fig.set_figheight(1)
plt=ConfusionMatrixDisplay(cnf3,display_labels=label)
plt.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fdef7dcf0d0>
```

]


551
279
66
87
1
0
True label
1
0
Predicted label
500
400
300
200
100

5. Random forest

```
from sklearn.ensemble import RandomForestClassifier
model4 = RandomForestClassifier()
model4.fit(x_train, y_train)
RandomForestClassifier()
y4_pred= model4.predict(x_test)
y4_pred

```

```
array([0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 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,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 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,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 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, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
```

```
[ ]
        0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
        1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
        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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
        0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0,
        0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
        1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0,
        0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
        0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
        1, 1, 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, 1, 0, 0, 0, 0, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0,
        1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0])
```

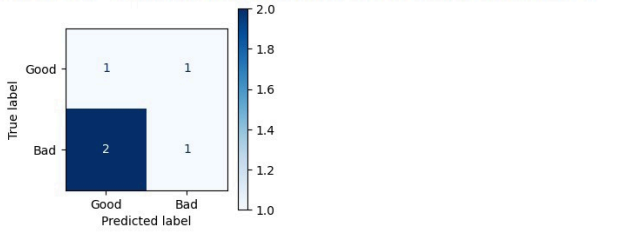
```
[ ]
1  from sklearn.metrics import confusion_matrix,accuracy_score,ConfusionMatrixDisplay
2  cnf4=confusion_matrix(y4_pred,y_test)
3  print(cnf4)
4  score4=accuracy_score(y4_pred,y_test)
5  print("accuracy_score is",score4)
6  report4=classification_report(y4_pred,y_test)
7  print(report4)
```

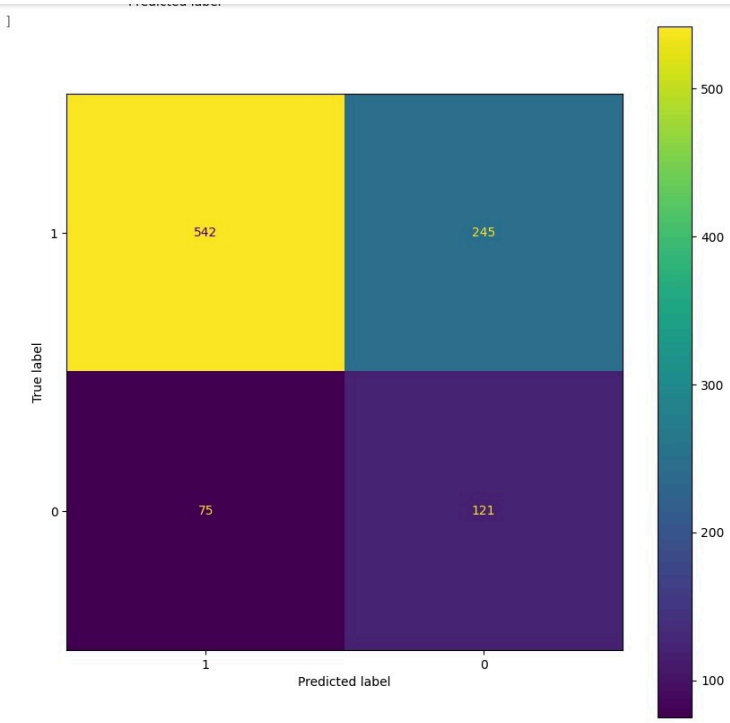
```
[[542 245]
 [ 75 121]]
accuracy_score is 0.6744659206510681
              precision    recall  f1-score   support

           0       0.88      0.69      0.77       787
           1       0.33      0.62      0.43       196

    accuracy                           0.67       983
   macro avg       0.60      0.65      0.60       983
weighted avg       0.77      0.67      0.70       983
```

```
[ ]
```

```python
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
label=[1,0]
disp = ConfusionMatrixDisplay.from_predictions(
                    [0,1,1,0,1],
                    [0,1,0,1,0],
                    cmap=plt.cm.Blues,
                    display_labels=['Good','Bad'],
                    values_format='',
)
fig = disp.ax_.get_figure()
fig.set_figwidth(3)
fig.set_figheight(3)
plt=ConfusionMatrixDisplay(cnf4,display_labels=label)
plt.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fdef4f81330>
```




542
245
75
121
1
0
True label
1
0
Predicted label
500
400
300
200
100

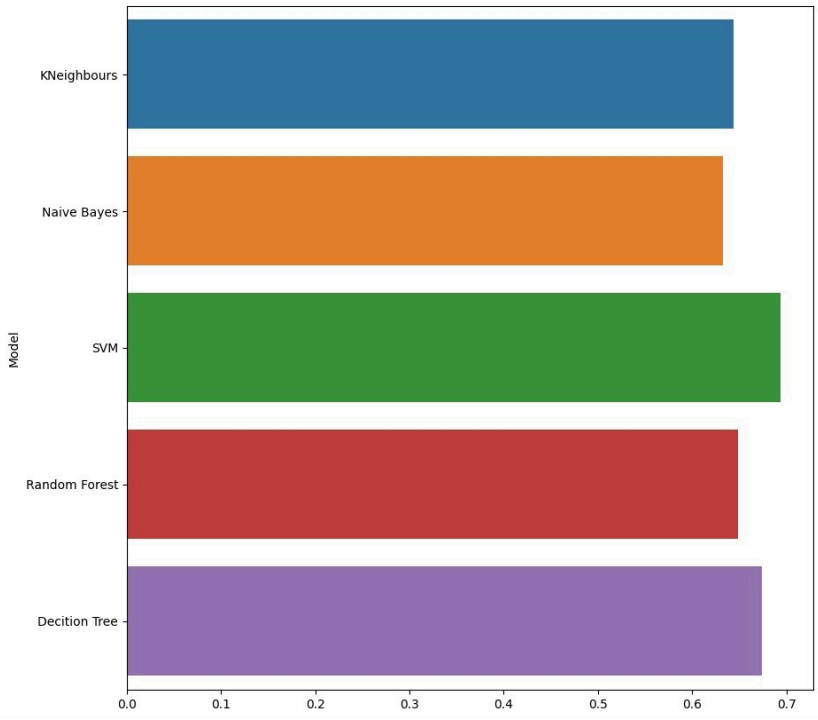
Comparison

```
[ ]
models = pd.DataFrame({
    'No:':[1,2,3,4,5],
    'Model':['KNeighbours','Naive Bayes','SVM','Random Forest','Decition Tree'],
    'Accuracy_score' :[score, score1,score2,score3,score4]
})
models
```

	No:	Model	Accuracy_score
0	1	KNeighbours	0.643947
1	2	Naive Bayes	0.632757
2	3	SVM	0.693795
3	4	Random Forest	0.649034
4	5	Decition Tree	0.674466

```
[ ]
sns.barplot(x='Accuracy_score', y='Model', data=models)
models.sort_values(by='Accuracy_score', ascending=False)
```

	No:	Model	Accuracy_score
2	3	SVM	0.693795
4	5	Decition Tree	0.674466
3	4	Random Forest	0.649034
0	1	KNeighbours	0.643947
1	2	Naive Bayes	0.632757


KNeighbours
Naive Bayes
SVM
Random Forest
Decition Tree
Model
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7