

## Water potability prediction

The quality of water needs to be examined in order to ascertain its safety to people.

The water\_potability.csv file contains water quality metrics for 3276 different water bodies.

```
ph-> pH of water
Hardness-> Capacity of water to precipitate soap in mg/L
Solids-> Total dissolved solids in ppm
Chloramines-> Amount of Chloramines in ppm
Sulfate-> Amount of Sulfates dissolved in mg/L
Conductivity-> Electrical conductivity of water in µS/cm
Organic_carbon-> Amount of organic carbon in ppm
Trihalomethanes-> Amount of Trihalomethanes in µg/L
Turbidity-> Measure of light emitting property of water in NTU      (Nephelometric Turbidity Units)
Potability-> Indicates if water is safe for human consumption
```

```
# 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.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

```
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

✓ 0s completed at 1:00 PM

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

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	🔗

print(df.shape)

(3276, 10)

print(df.columns)

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

# Describe the dataset which shows the minimum value, maximum value, mean value, count, standard deviation, etc.  
df.describe()

	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	🔗

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
```

```
# Checking for missing values
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
```

```
# Filling the missing values using a mean value of each feature
df['ph']=df['ph'].fillna(df['ph'].mean())
df['Sulfate']=df['Sulfate'].fillna(df['Sulfate'].mean())
df['Trihalomethanes']=df['Trihalomethanes'].fillna(df['Trihalomethanes'].mean())
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
```

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

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

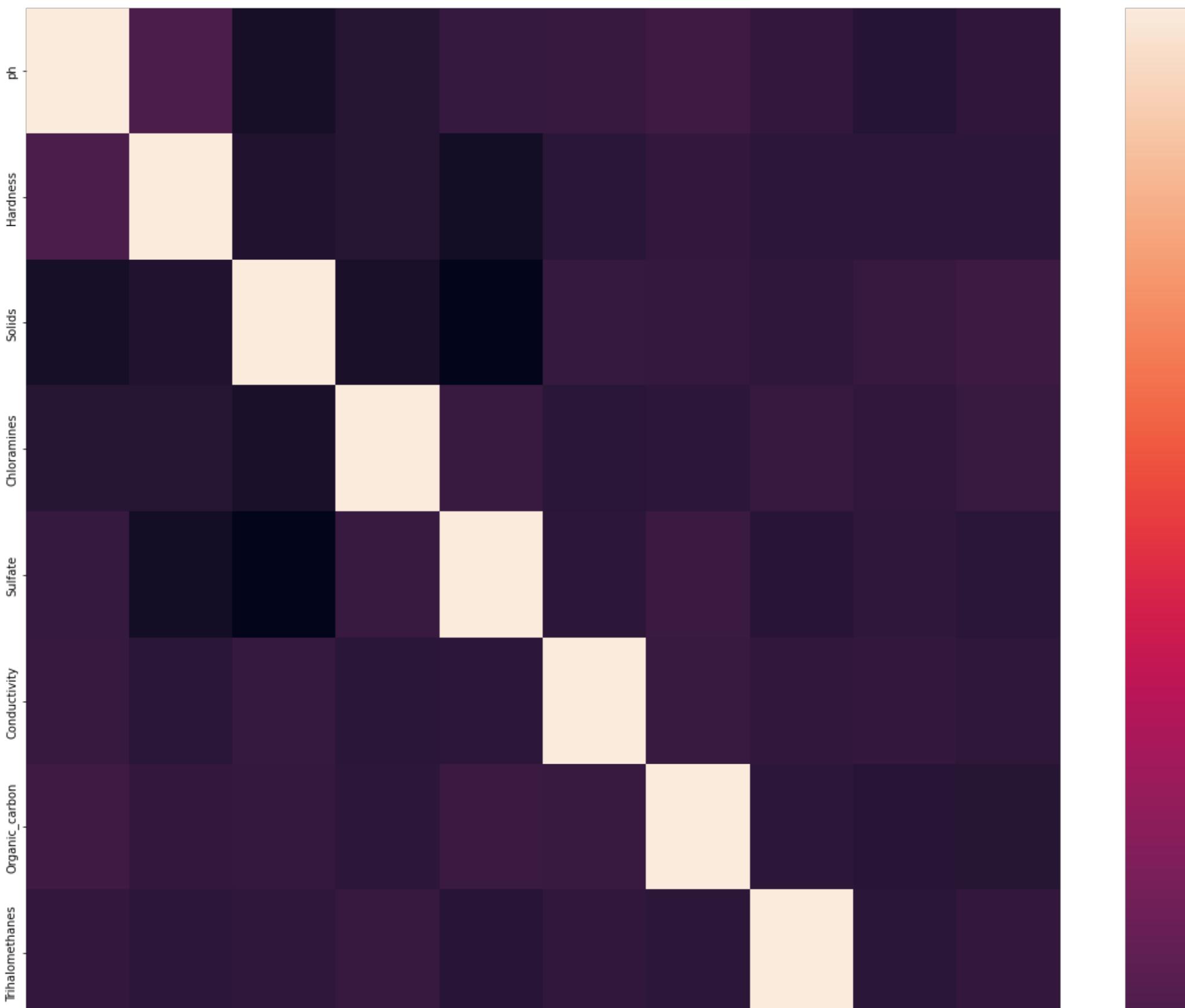
```
df.corr() #find the correlation
```

	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	

<b>Conductivity</b>	0.017192	-0.023915	0.013831	-0.020486	-0.014059	1.000000	0.020966	0.001255	0.005798	-0.008128
<b>Organic_carbon</b>	0.040061	0.003610	0.010242	-0.012653	0.026909	0.020966	1.000000	-0.012976	-0.027308	-0.030001
<b>Trihalomethanes</b>	0.002994	-0.012690	-0.008875	0.016627	-0.025605	0.001255	-0.012976	1.000000	-0.021502	0.006960
<b>Turbidity</b>	-0.036222	-0.014449	0.019546	0.002363	-0.009790	0.005798	-0.027308	-0.021502	1.000000	0.001581
<b>Potability</b>	-0.003287	-0.013837	0.033743	0.023779	-0.020619	-0.008128	-0.030001	0.006960	0.001581	1.000000

```
sns.heatmap(df.corr())
```

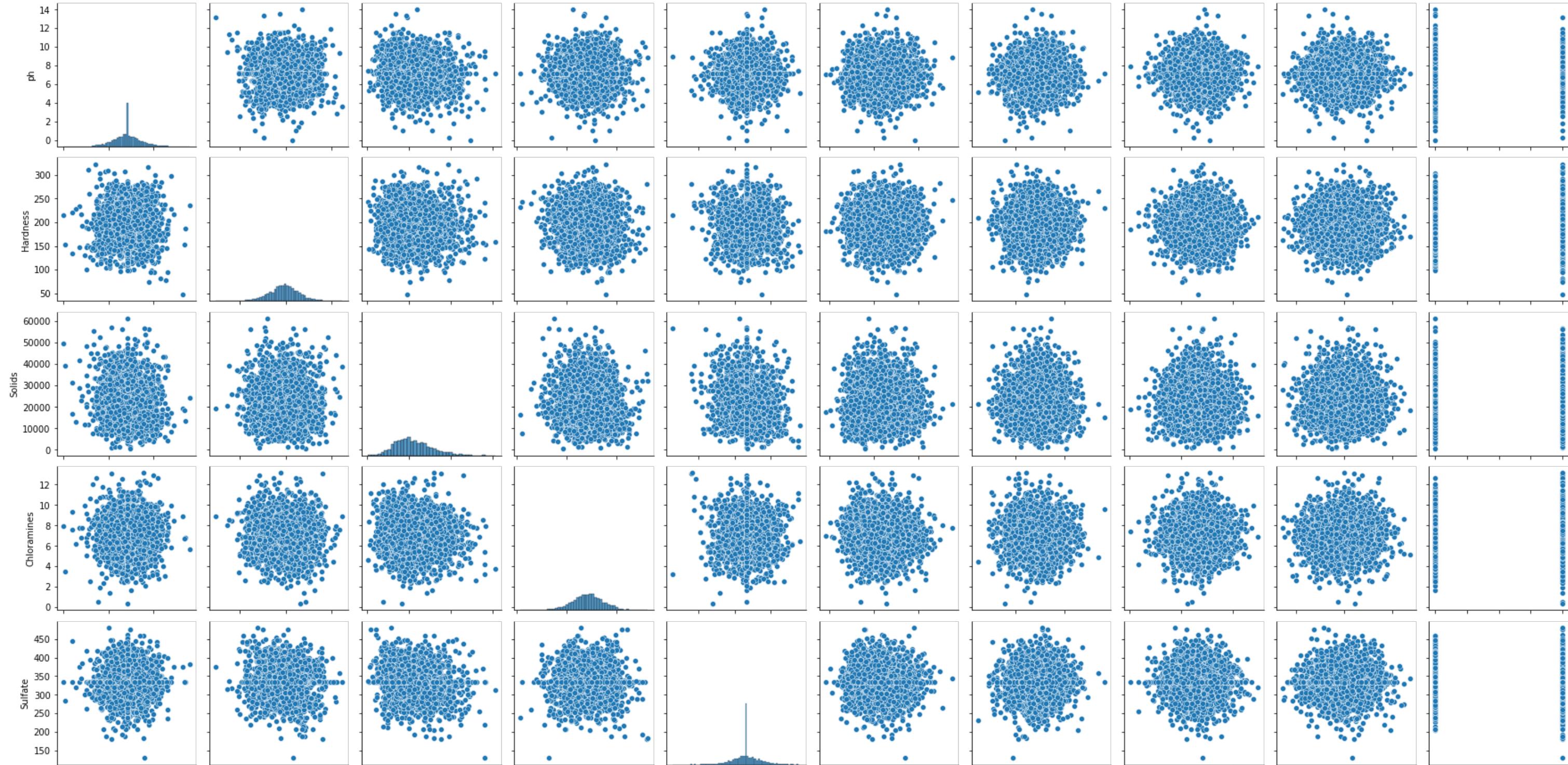
<matplotlib.axes.\_subplots.AxesSubplot at 0x7f60b2440e80>

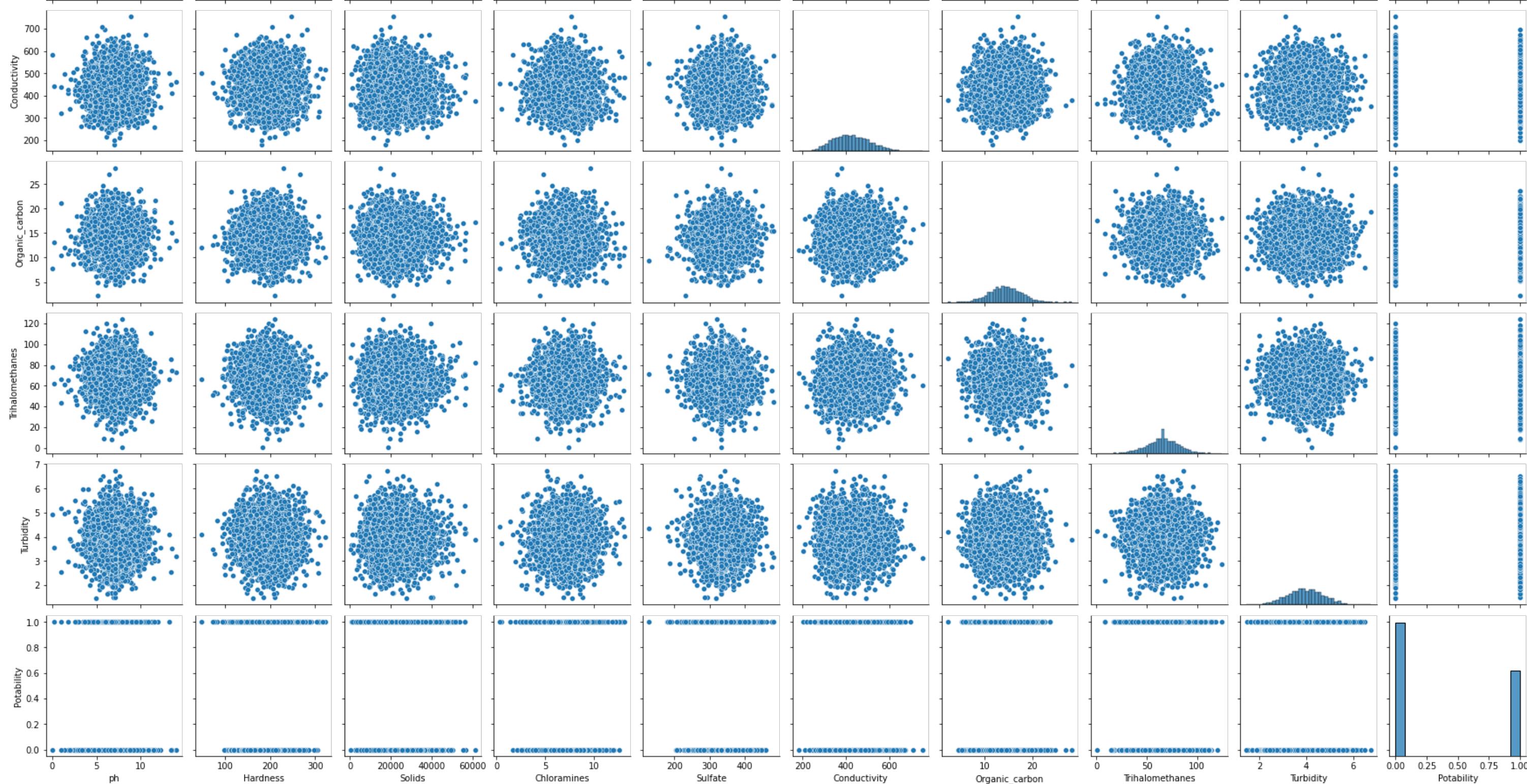




```
sns.pairplot(df)
```

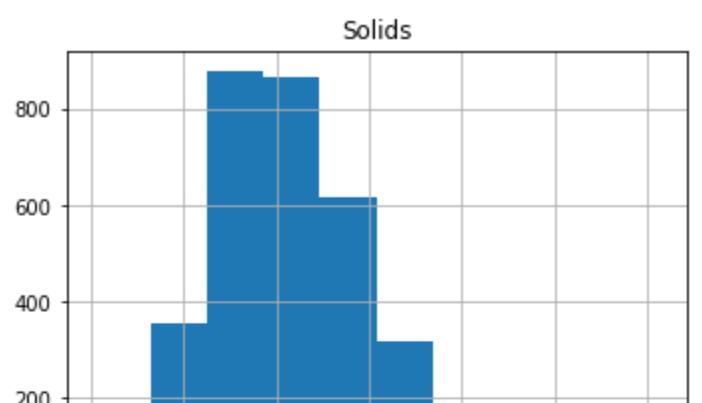
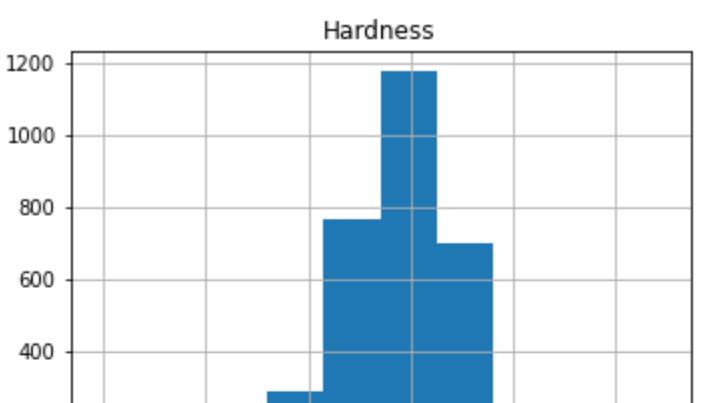
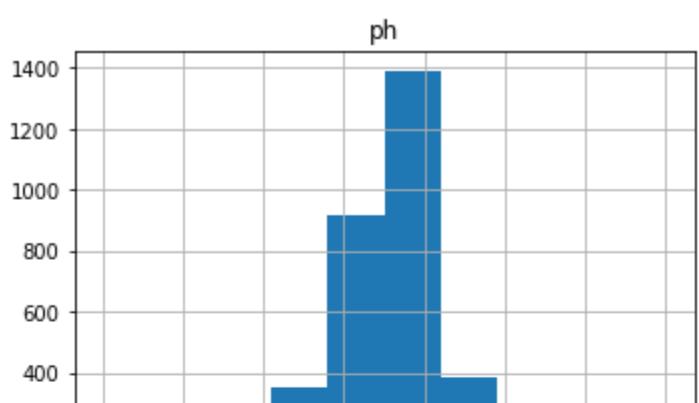
<seaborn.axisgrid.PairGrid at 0x7f60b738d730>

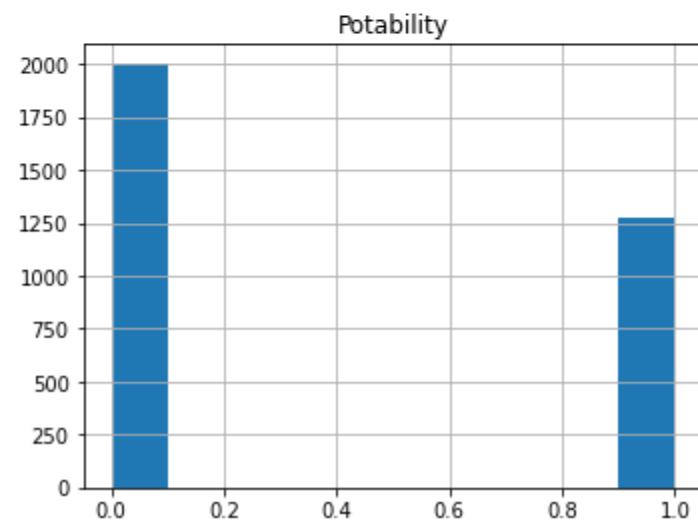
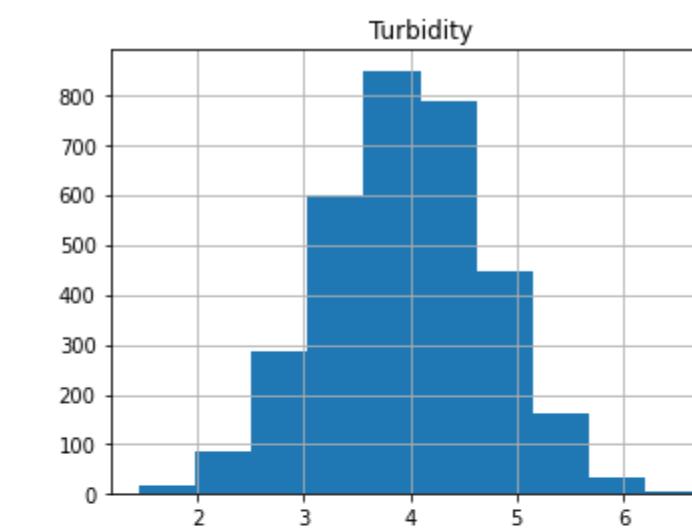
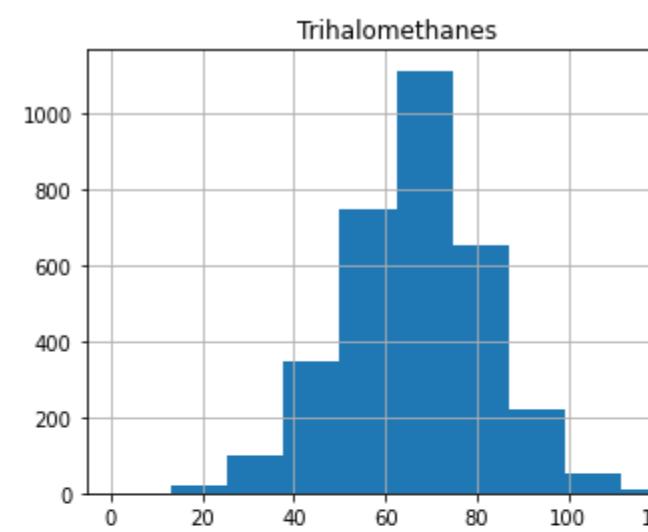
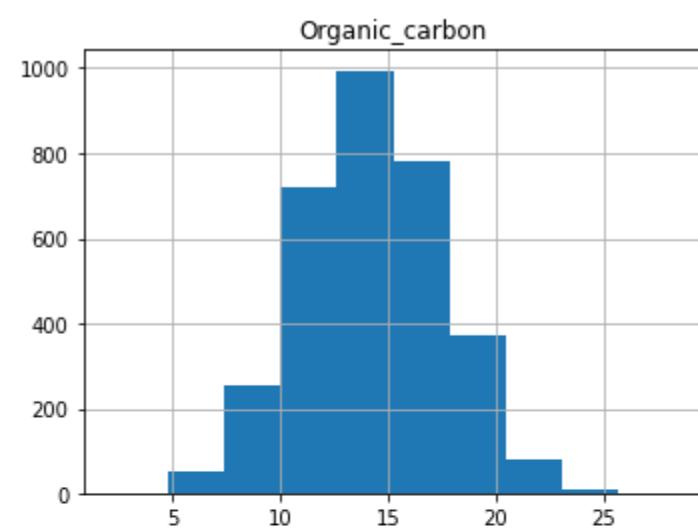
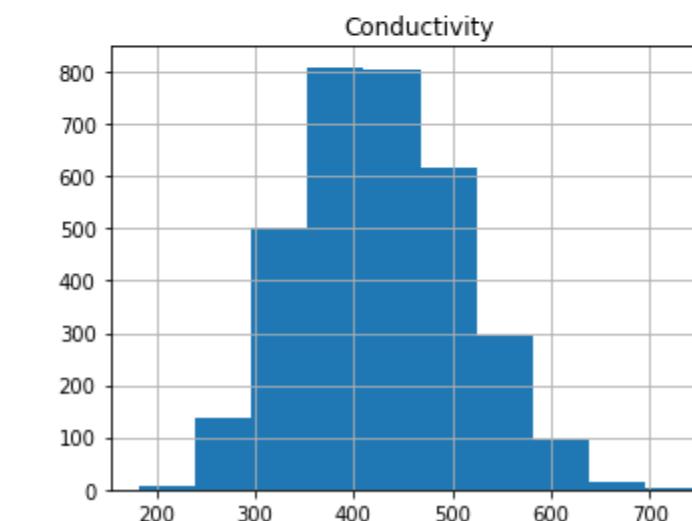
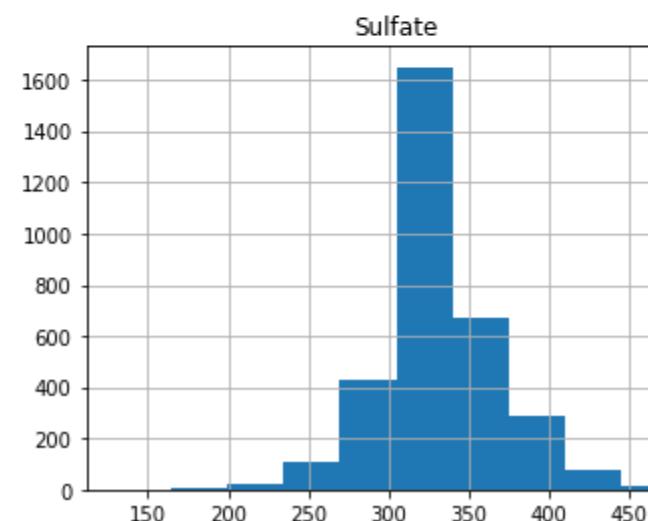
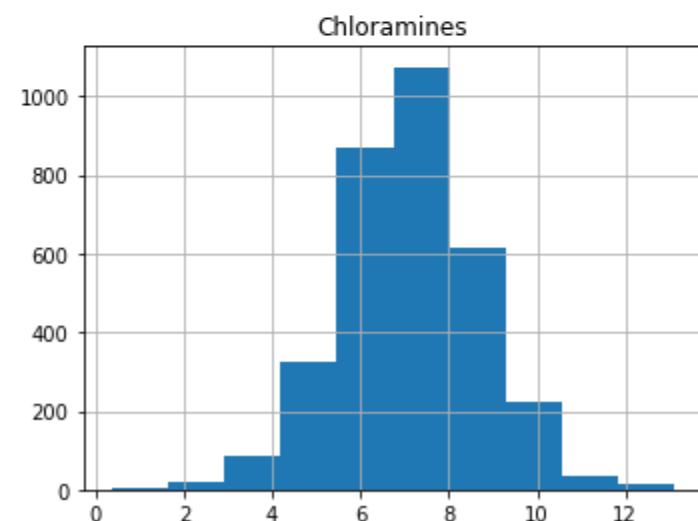
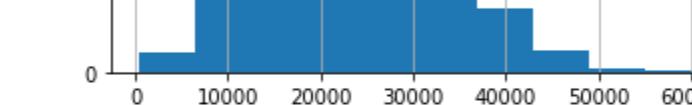
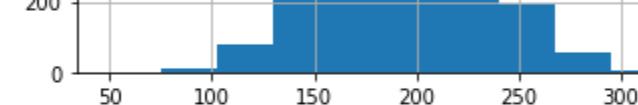
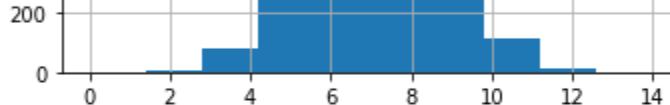




```

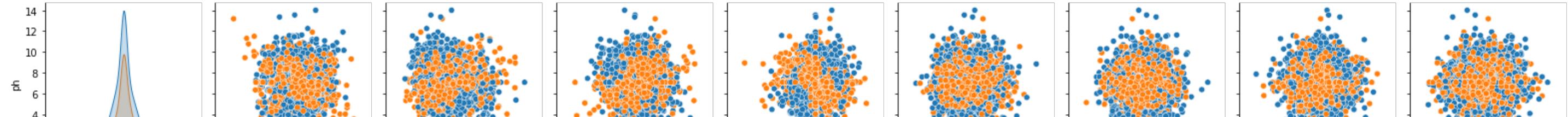
plt.rcParams['figure.figsize'] = [20,20]
df.hist()
plt.show()
# defines a runtime configuration (rc) containing the default styles for every plot element you create
    
```

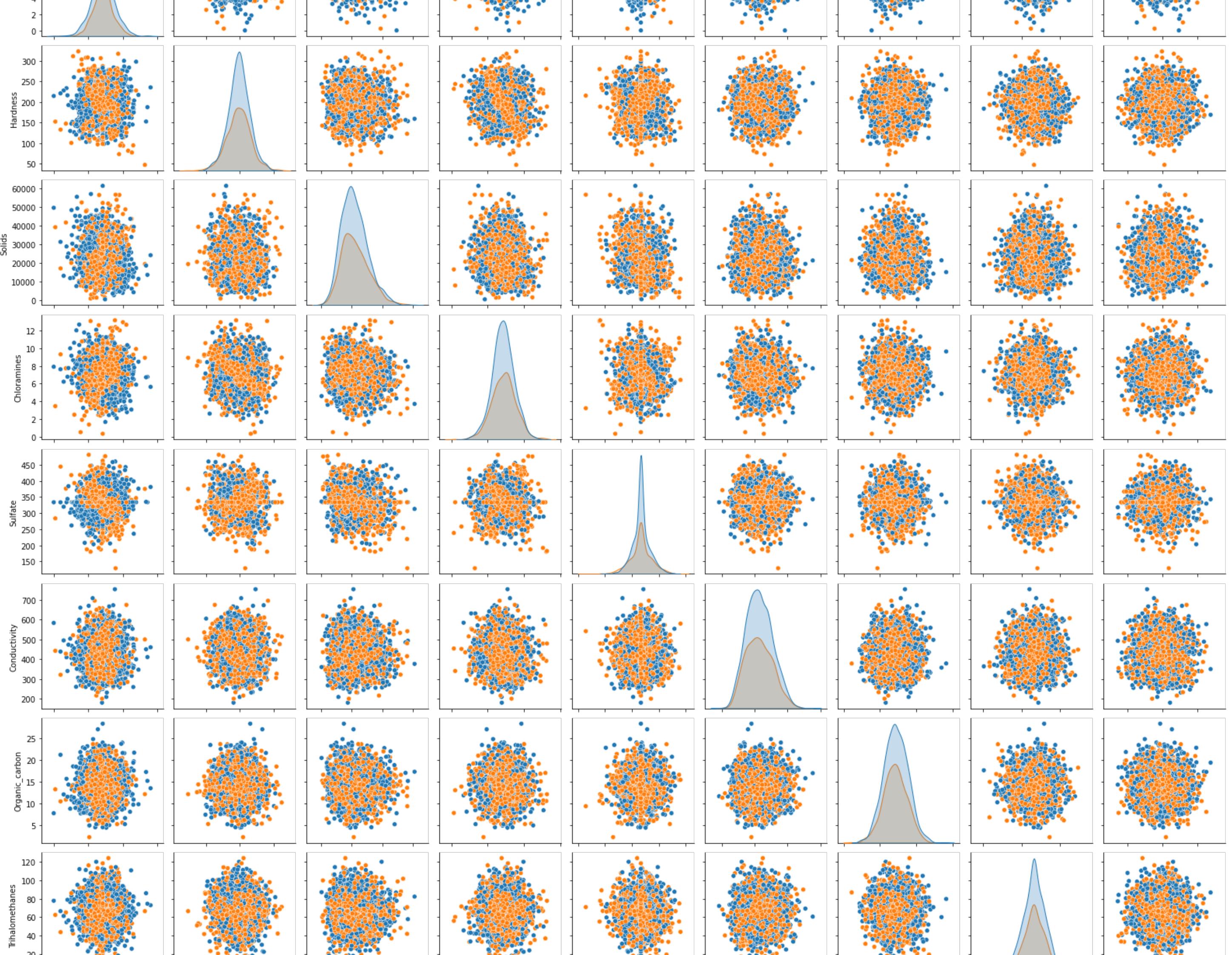


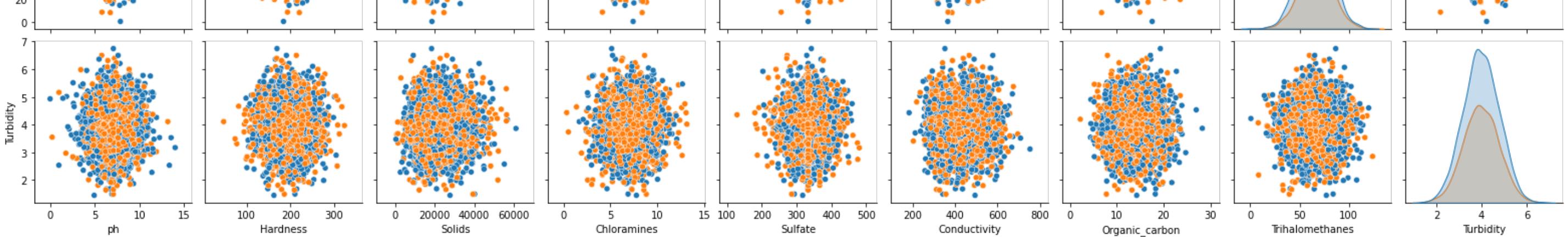


```
sns.pairplot(df, hue="Potability")
```

```
<seaborn.axisgrid.PairGrid at 0x7f60b05532b0>
```



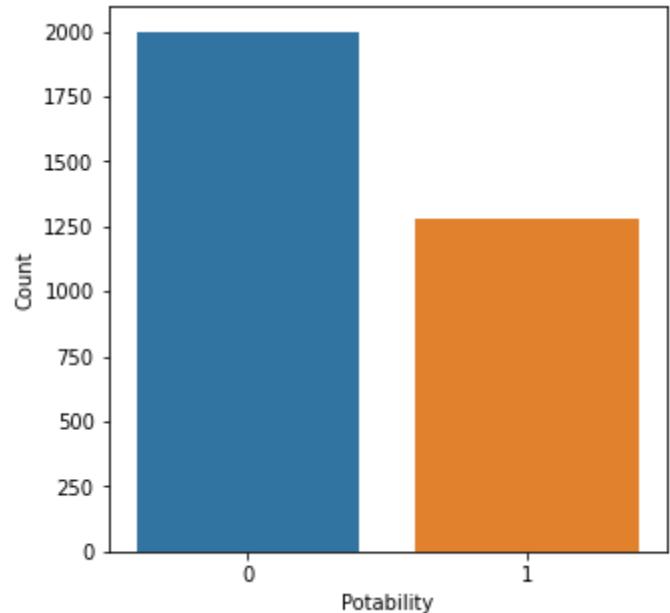




```
# Visualising the potability using a countplot function.
```

```
plt.figure(figsize=(5,5))
sns.countplot(df['Potability'])
plt.xlabel('Potability', fontsize=10)
plt.ylabel('Count', fontsize=10)
plt.show()
```

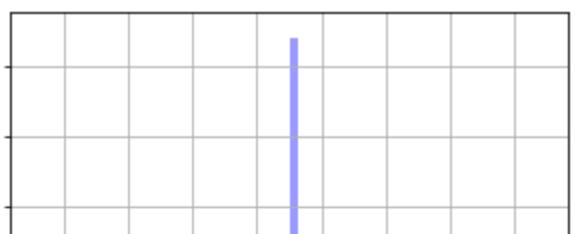
/usr/local/lib/python3.8/dist-packages/seaborn/\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be "color". To future-proof your code, please pass the keyword argument instead. To turn off this warning, use warnings.simplefilter("ignore").

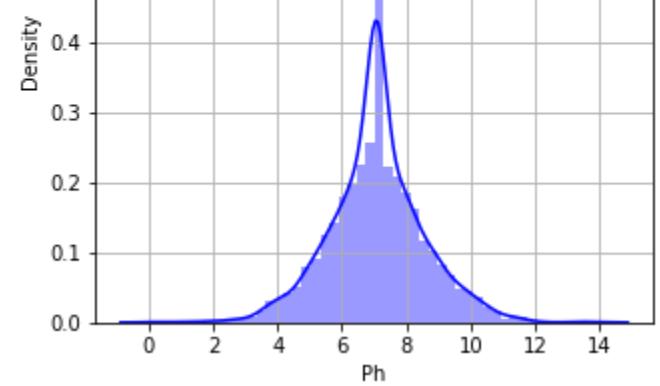


```
# visualising the pH value using a distplot function.
```

```
plt.figure(figsize=(5,5))
sns.distplot(df['ph'], color='blue')
plt.xlabel('Ph', fontsize=10)
plt.ylabel('Density', fontsize=10)
plt.grid()
plt.show()
```

/usr/local/lib/python3.8/dist-packages/seaborn/distributions.py:2619: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` or `kdeplot`. To turn off this warning, use warnings.simplefilter("ignore").



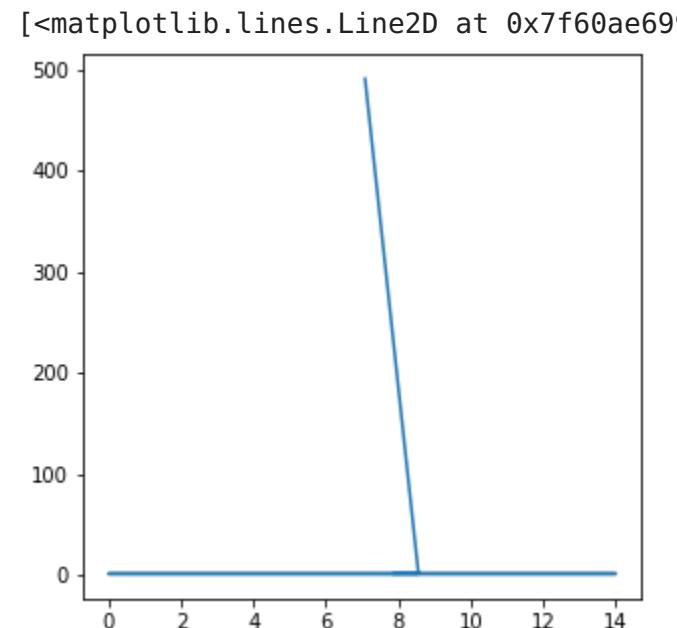


Find the value count and plot

```
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=(5,5))  
plt.plot(ph)
```



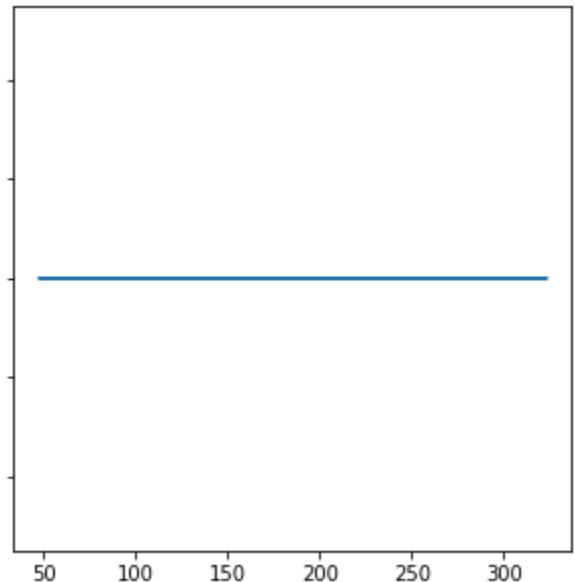
```
Hardness=df['Hardness'].value_counts()  
Hardness
```

```
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
```

```
plt.figure(figsize=(5,5))  
plt.plot(Hardness)
```

```
[<matplotlib.lines.Line2D at 0x7f60ae666be0>]
```



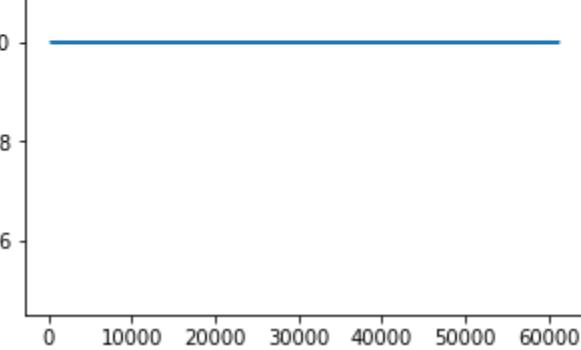
```
Solids=df['Solids'].value_counts()  
Solids
```

```
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=(5,5))  
plt.plot(Solids)
```

```
[<matplotlib.lines.Line2D at 0x7f60ae63b730>]
```

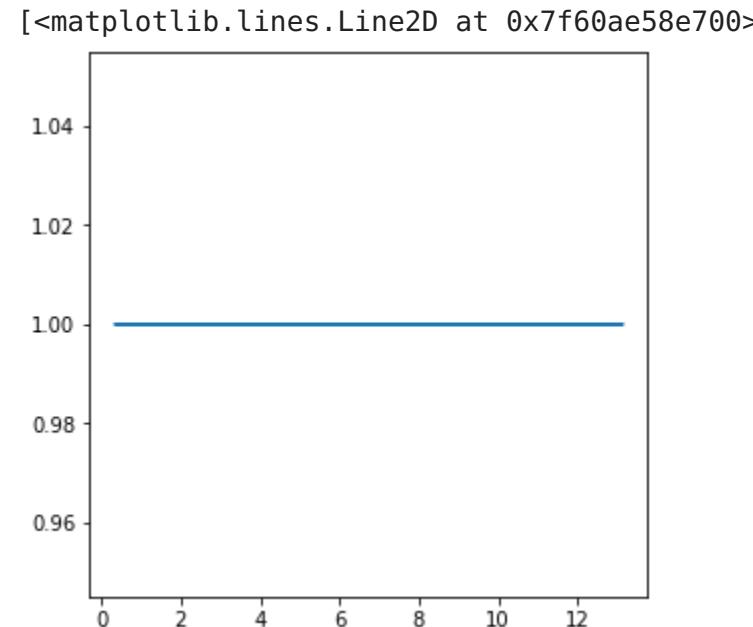




```
Chloramines=df['Chloramines'].value_counts()  
Chloramines
```

```
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=(5,5))  
plt.plot(Chloramines)
```

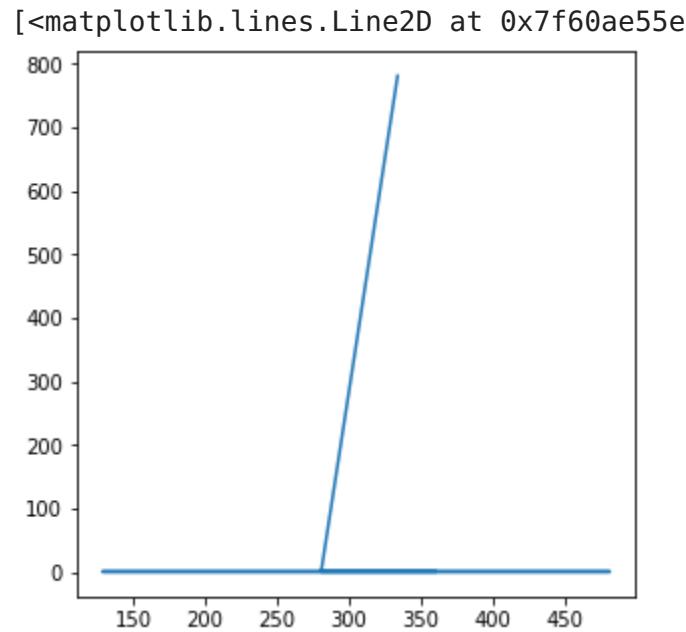


```
Sulfate=df['Sulfate'].value_counts()  
Sulfate
```

```
333.775777    781  
280.745623    1  
332.744519    1  
391.918229    1  
330.905370    1  
.  
343.620823    1  
359.710517    1  
389.219586    1
```

```
389.215586    1  
337.231469    1  
359.948574    1  
Name: Sulfate, Length: 2496, dtype: int64
```

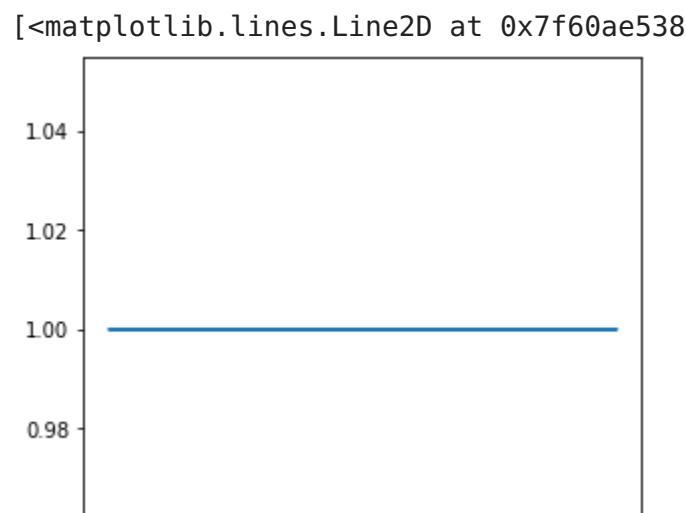
```
plt.figure(figsize=(5,5))  
plt.plot(Sulfate)
```

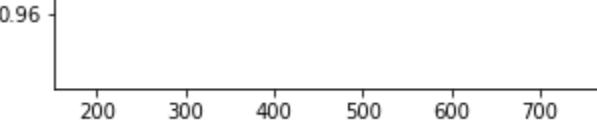


```
Conductivity=df['Conductivity'].value_counts()  
Conductivity
```

```
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
```

```
plt.figure(figsize=(5,5))  
plt.plot(Conductivity)
```



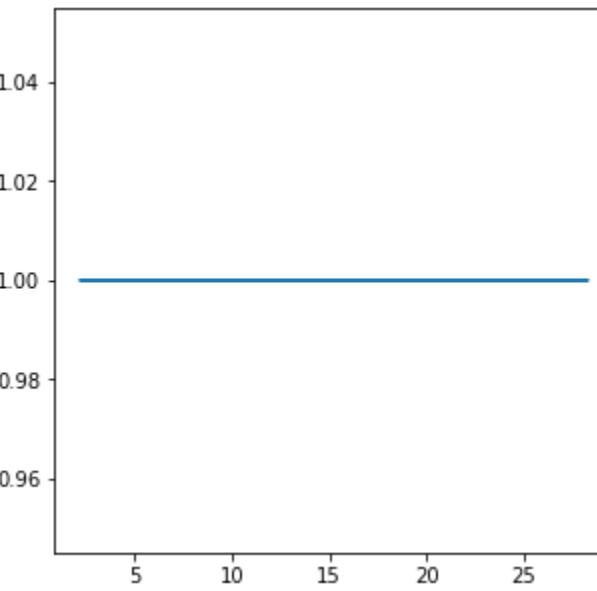


```
Organic_carbon=df['Organic_carbon'].value_counts()  
Organic_carbon
```

```
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
```

```
plt.figure(figsize=(5,5))  
plt.plot(Organic_carbon)
```

```
[<matplotlib.lines.Line2D at 0x7f60ae48c5b0>]
```

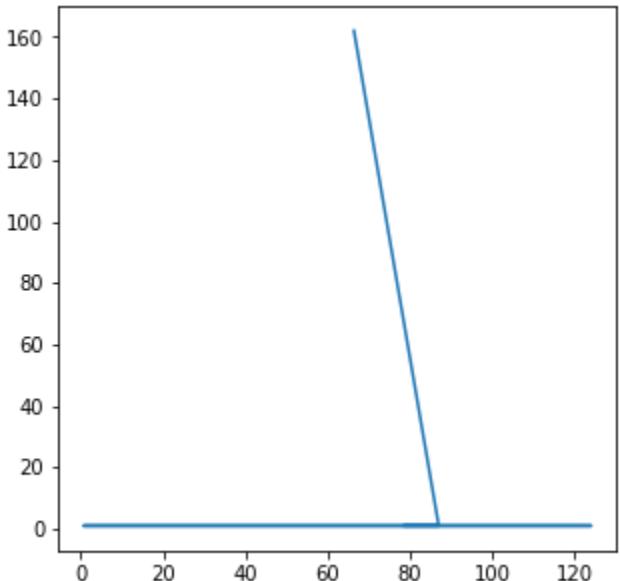


```
Trihalomethanes=df['Trihalomethanes'].value_counts()  
Trihalomethanes
```

```
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
```

```
plt.figure(figsize=(5,5))
plt.plot(Trihalomethanes)
```

```
[<matplotlib.lines.Line2D at 0x7f60ae454be0>]
```



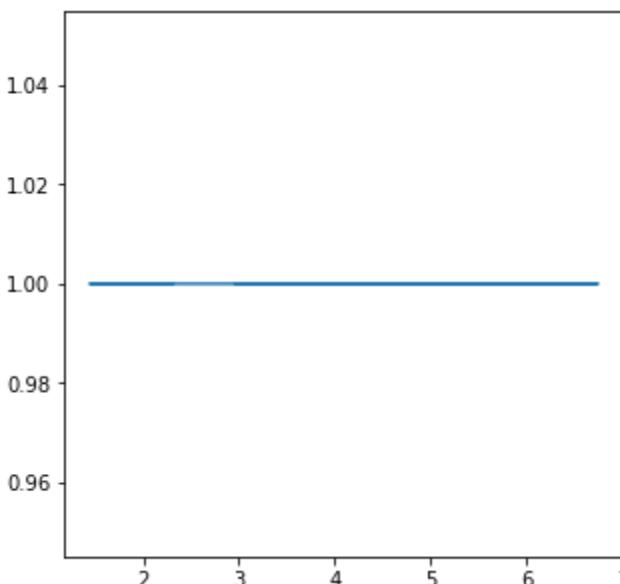
```
Turbidity=df['Turbidity'].value_counts()
```

```
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
```

```
plt.figure(figsize=(5,5))
plt.plot(Turbidity)
```

```
[<matplotlib.lines.Line2D at 0x7f60ae42fee0>]
```



```
# to print the datatypes  
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
```

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

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

## Normalization

```
from sklearn.preprocessing import StandardScaler  
scaler=StandardScaler()  
scaler.fit(x_train)  
x_train=scaler.transform(x_train)  
x_test=scaler.transform(x_test)  
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 notebook.

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(cv=10, estimator=KNeighborsClassifier(),  
            param_grid={'n_neighbors': [3, 5, 7, 9, 8, 6, 11, 10],  
                        'weights': ['uniform', 'distance']},  
            scoring='accuracy')
```

```
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, 0, 1, 0, 1, 0,  
1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 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, 0,  
0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,  
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, 0, 0, 0, 0, 1, 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, 1, 1, 1, 1, 0, 1,
1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 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, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1,
1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 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, 1, 0, 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, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 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, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 1, 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, 0, 0, 0, 0, 0,
0, 1, 1, 1, 0, 0, 0, 1, 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, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
1, 1, 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, 0, 0, 0, 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, 1, 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, 1, 0, 1, 1, 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, 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, 1, 0, 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, 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, 1, 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, 1, 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, 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)
score
report=classification_report(y_pred,y_test)
print(report)

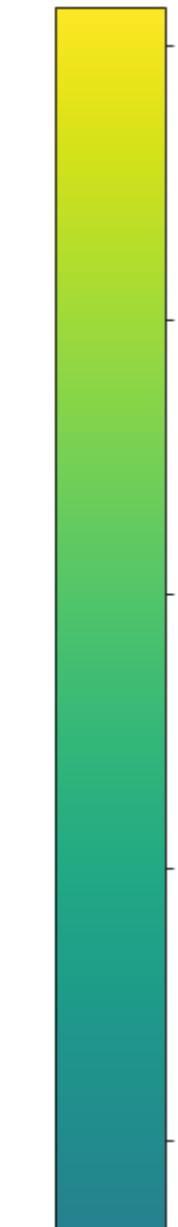
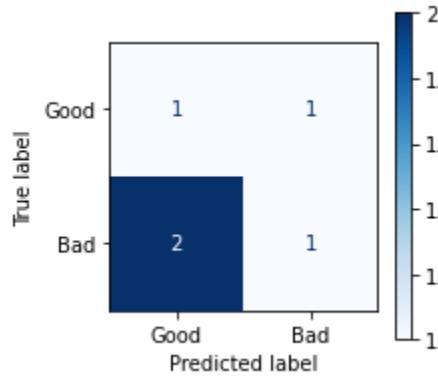
```

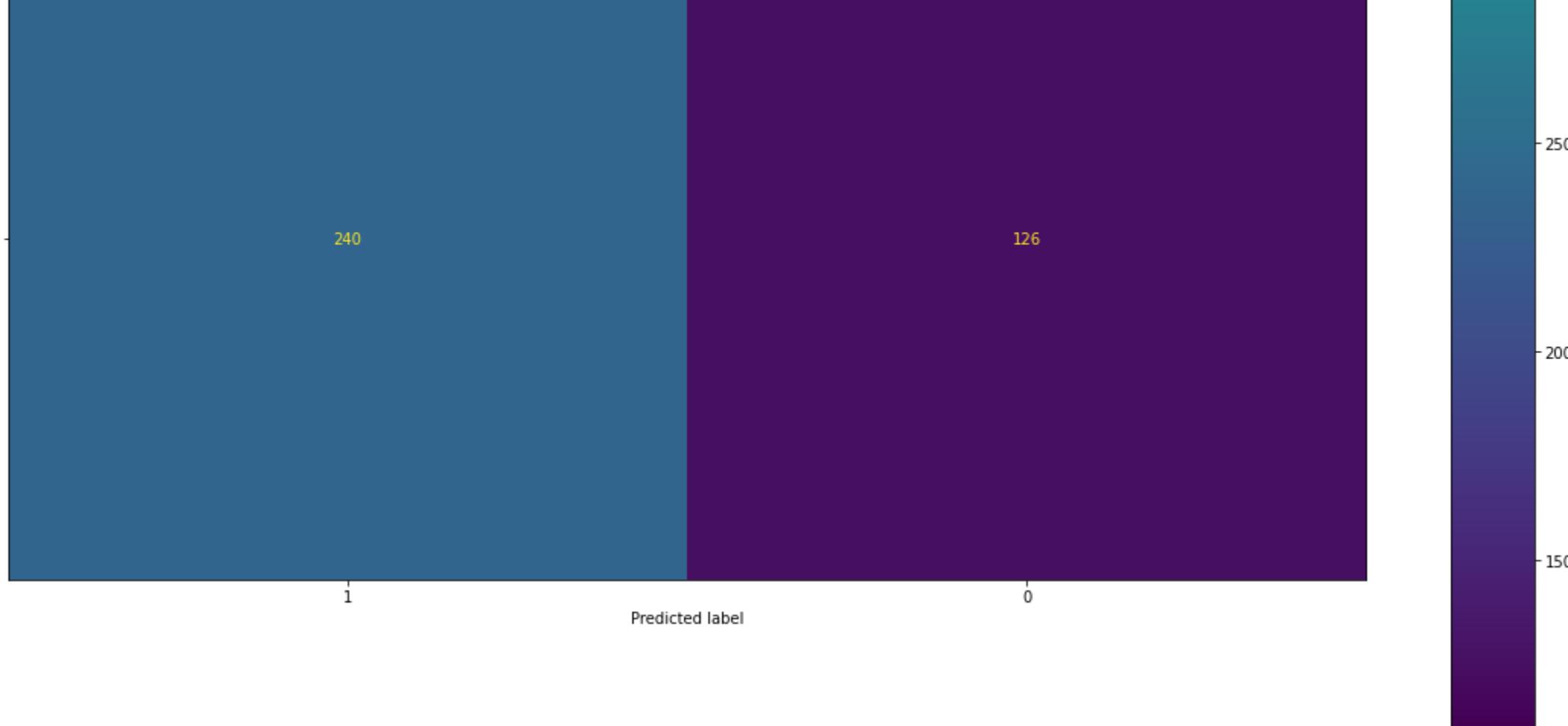
	precision	recall	f1-score	support
0	0.82	0.68	0.74	747
1	0.34	0.53	0.42	236
accuracy			0.64	983
macro avg	0.58	0.61	0.58	983
weighted avg	0.71	0.64	0.67	983

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

```
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(cnf,display_labels=label)
plt.plot()
```

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x7f60ae28e640>





## 2. Naive Bayes Algorithm

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

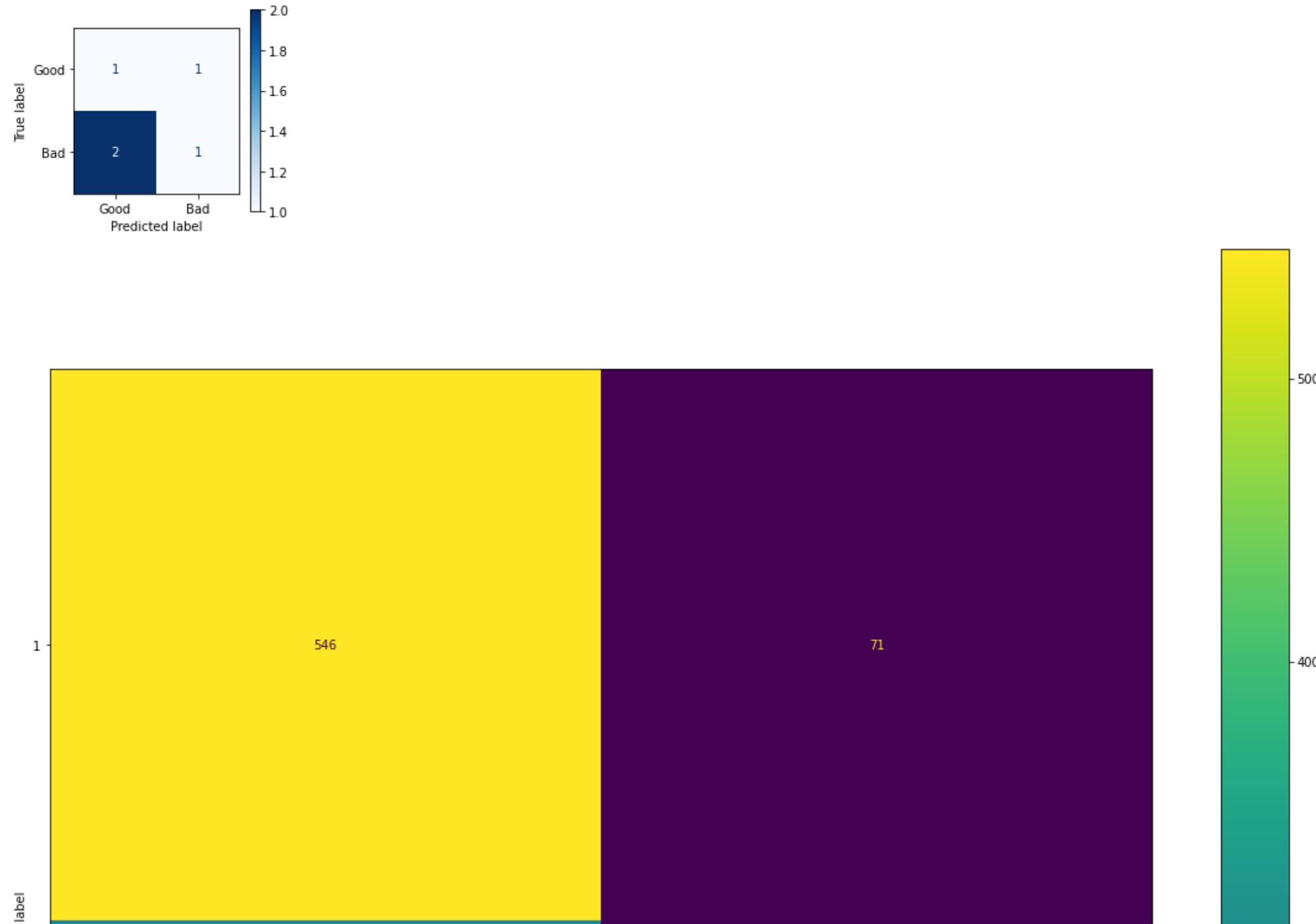
[1]
```

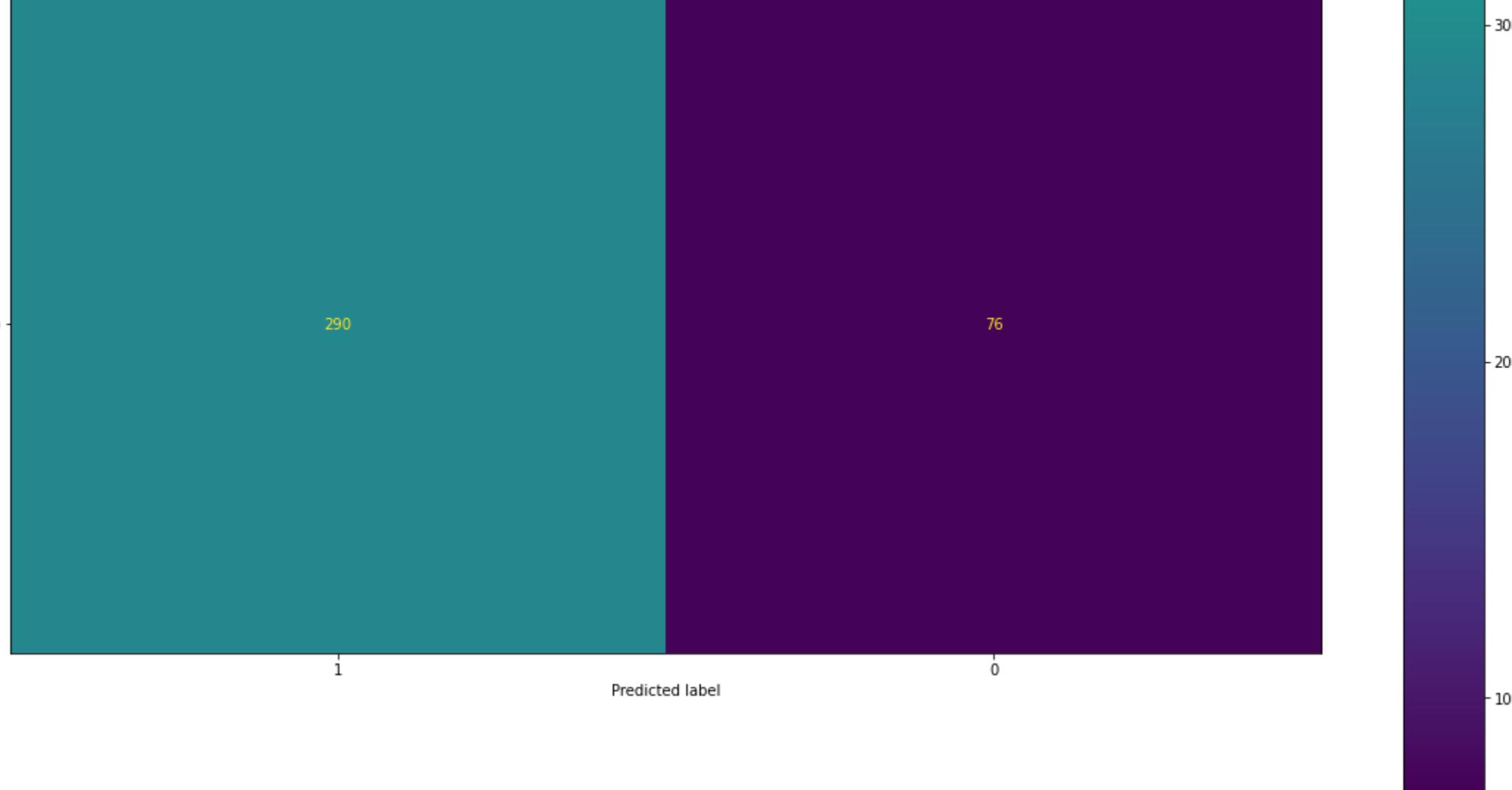
```
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report,ConfusionMatrixDisplay
cnf1=confusion_matrix(y_test,y1_pred)
print(cnf1)
score1=accuracy_score(y1_pred,y_test)
score1
report1=classification_report(y1_pred,y_test)
print(report1)
```

[[546 71]	
[290 76]]	
	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

```
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(cnfl,display_labels=label)
plt.plot()
```

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x7f60ae1e4f40>





### 3. Support Vector Machine Algorithm

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

[1]

```
from sklearn.metrics import confusion_matrix,accuracy_score
cnf2=confusion_matrix(y_test,y2_pred)
print(cnf2)
score2=accuracy_score(y2_pred,y_test)
score2
report2=classification_report(y2_pred,y_test)
print(report2)
```

`[[573 44]  
 [257 109]]`

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

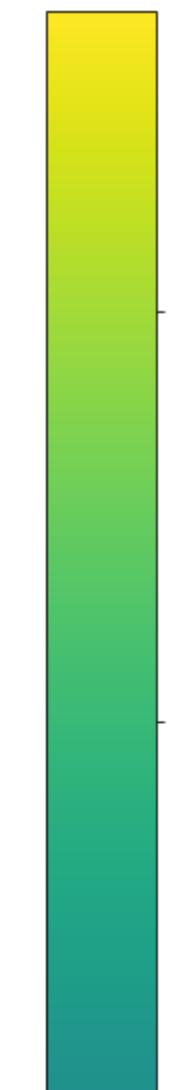
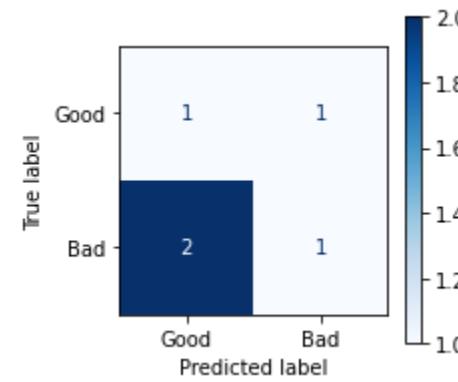
0	0.93	0.69	0.79	830
1	0.30	0.71	0.42	153

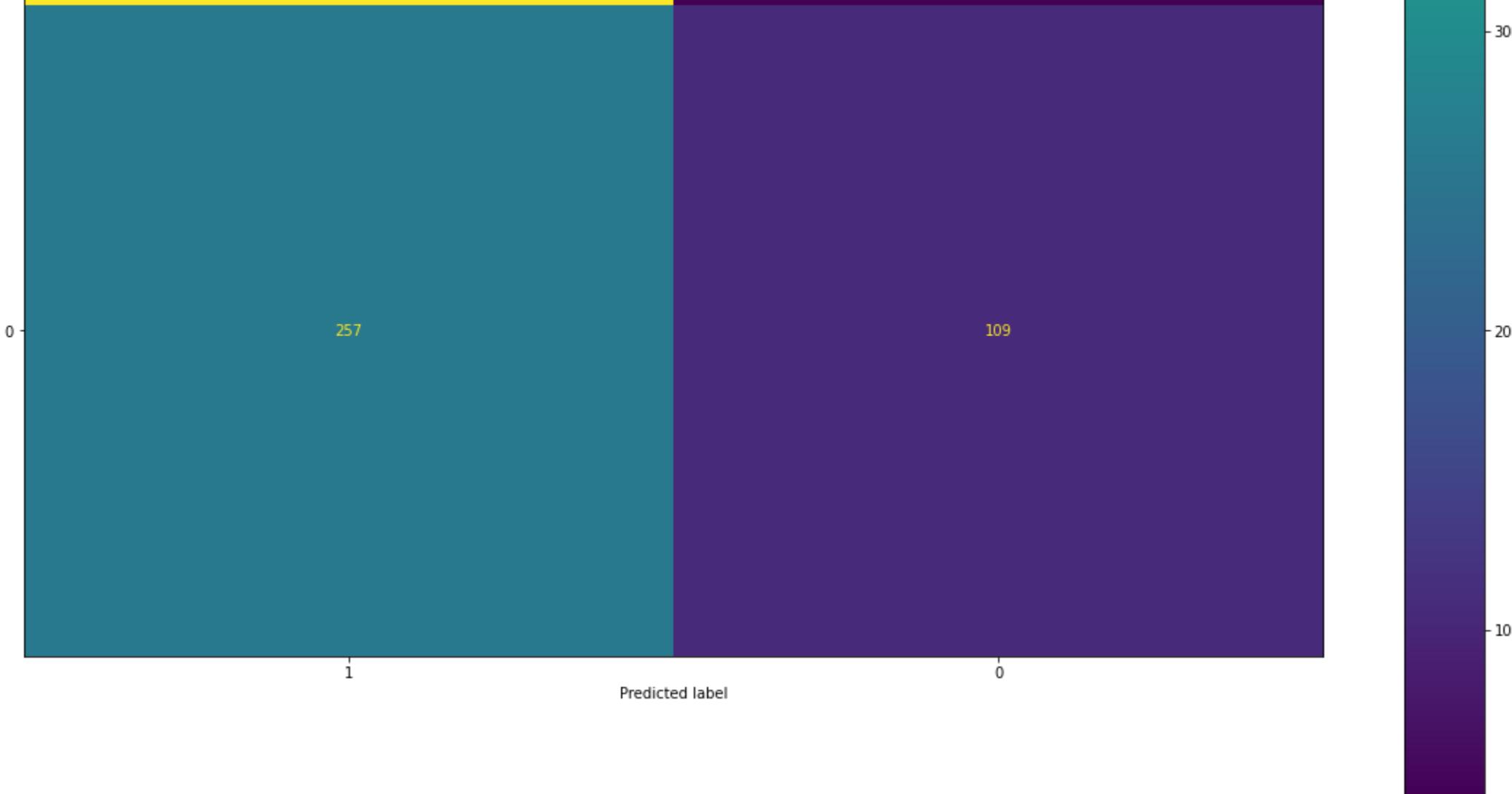
accuracy		0.69	0.69	983
macro avg	0.61	0.70	0.61	983
weighted avg	0.83	0.69	0.73	983

weighted avg 0.65 0.69 0.73 983

```
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(3)
fig.set_figheight(3)
plt=ConfusionMatrixDisplay(cnf2,display_labels=label)
plt.plot()
```

<sklearn.metrics.\_plot.confusion\_matrix.ConfusionMatrixDisplay at 0x7f60ae0881c0>





## 4. Decision Tree Classifier

```

1, 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, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
0, 1, 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, 0, 1, 0, 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, 1, 0, 0, 0, 1, 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, 1,
1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 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, 1, 0, 0, 1, 0, 0, 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, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 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, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 1, 0, 1, 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, 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, 1, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 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, 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, 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]

```

```

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)
score3
print("accuracy_score is",score)
report3=classification_report(y3_pred,y_test)
print(report3)

```

```

[[551 279]
 [ 66  87]]
accuracy_score is 0.6439471007121058
      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

```

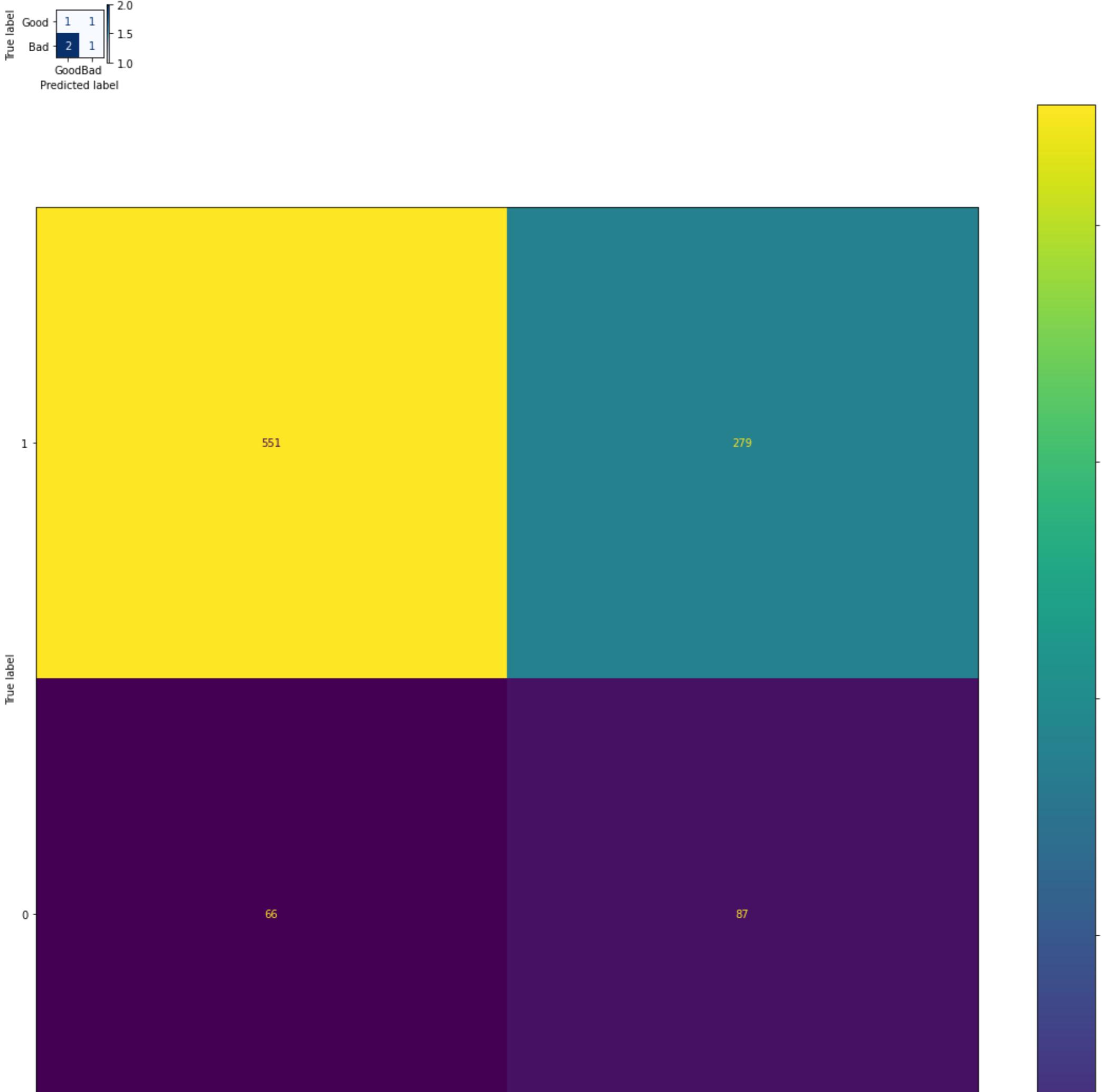
```

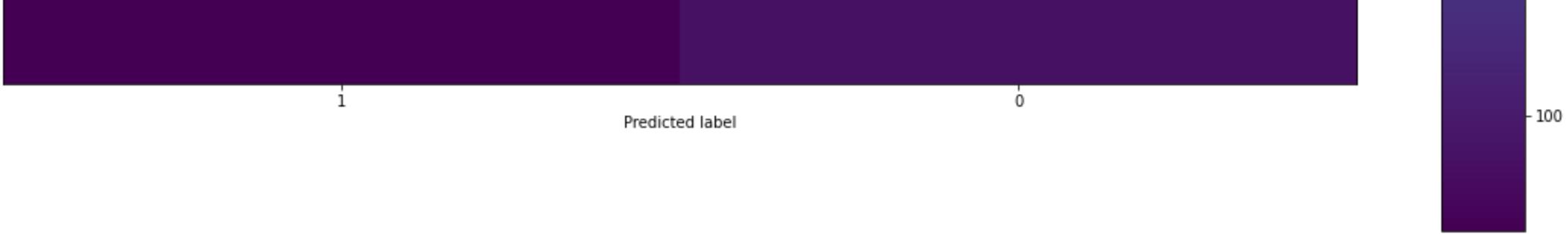
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
label=[1,0]
disp = ConfusionMatrixDisplay.from_predictions(
    [0,1,1,0,1],
    [0,1,0,1,0],
    cmap=plt.cm.Blues,
    display_labels=['Good', 'Bad'],
    values_format='',
)
fig = disp.ax_.get_figure()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

```

```
fig.set_tight_layout(1)
plt=ConfusionMatrixDisplay(cnf3,display_labels=label)
plt.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f60adf89e80>
```





## 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
```

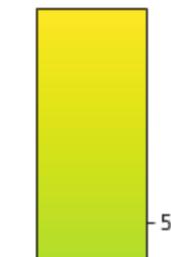
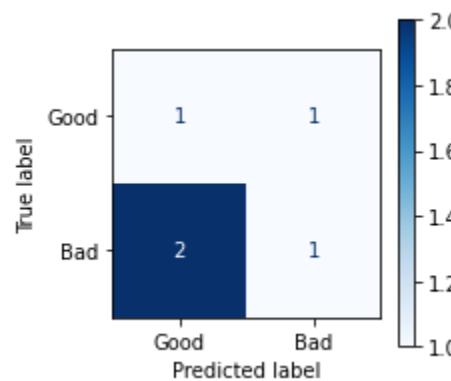
```
1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,  
0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,  
0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0,  
1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

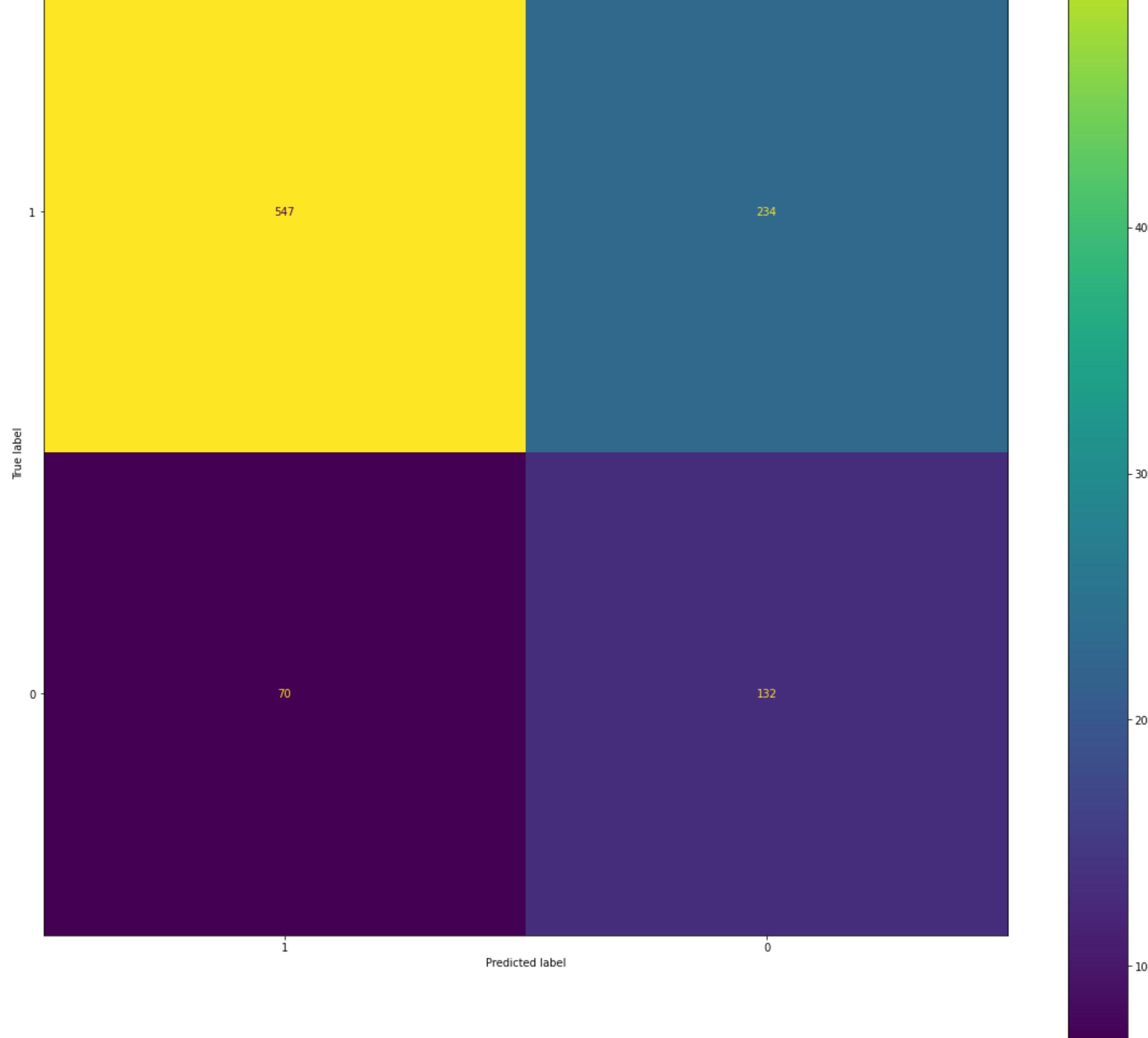
```
from sklearn.metrics import confusion_matrix, accuracy_score, ConfusionMatrixDisplay  
cnf4=confusion_matrix(y4_pred,y_test)  
cnf4  
score4=accuracy_score(y4_pred,y_test)  
score4  
print("accuracy_score is",score4)  
report4=classification_report(y4_pred,y_test)  
print(report4)
```

```
accuracy_score is 0.6439471007121058  
precision recall f1-score support  
  
0 0.89 0.70 0.78 781  
1 0.36 0.65 0.46 202  
  
accuracy 0.69 983  
macro avg 0.62 0.68 0.62 983  
weighted avg 0.78 0.69 0.72 983
```

```
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(3)  
fig.set_figheight(3)  
plt=ConfusionMatrixDisplay(cnf4,display_labels=label)  
plt.plot()
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7f60ae4d8a90>
```





## Comparison

```
models = pd.DataFrame({  
    'Model': ['KNeighbours', 'Naive Bayes', 'SVM', 'Random Forest', 'Decision Tree'],  
    'Accuracy_score': [score, score1, score2, score3, score4]  
})
```

```
})  
models
```

Model	Accuracy_score
-------	----------------

0	KNeighbours	0.643947
1	Naive Bayes	0.632757
2	SVM	0.693795
3	Random Forest	0.649034
4	Decition Tree	0.690743



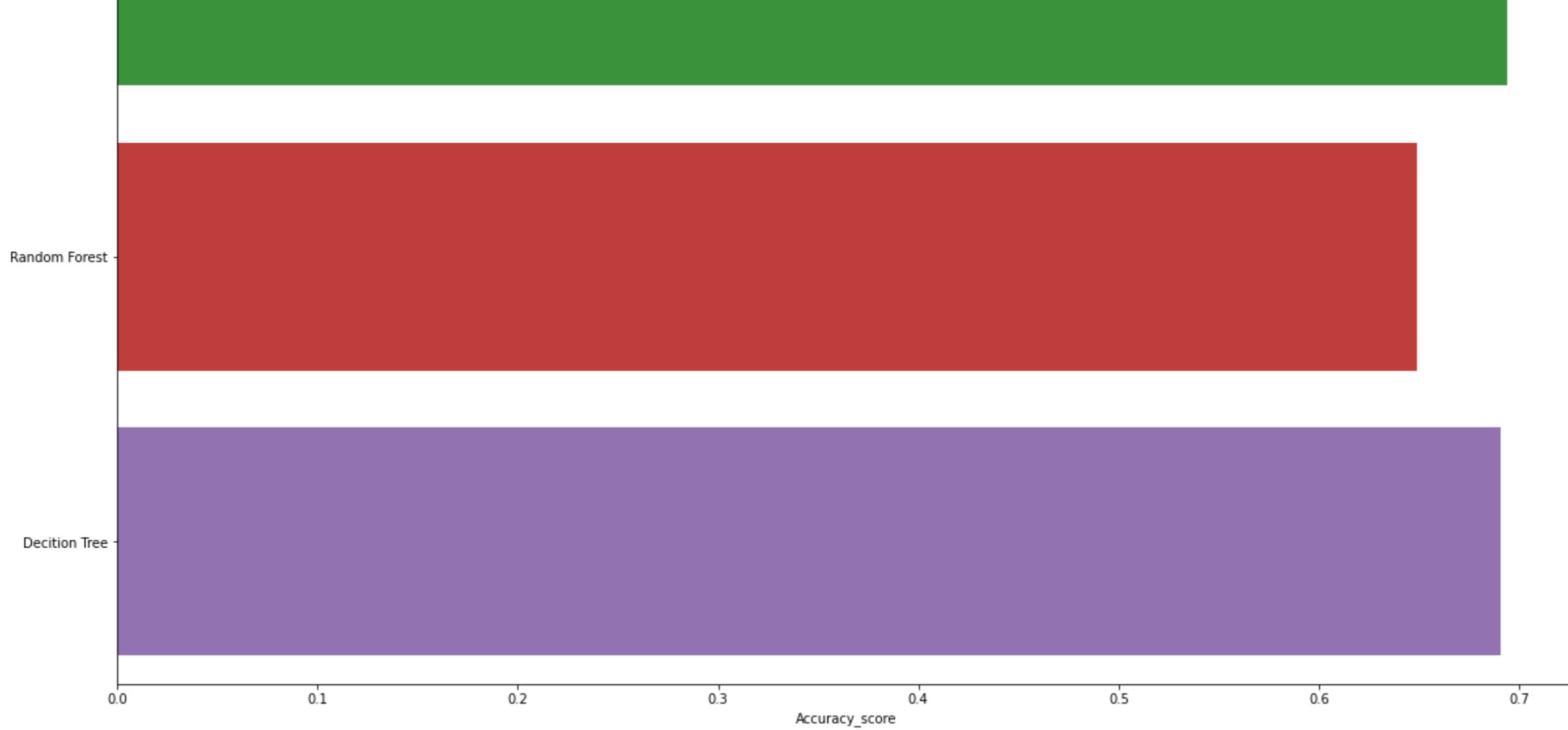
```
sns.barplot(x='Accuracy_score', y='Model', data=models)
```

```
models.sort_values(by='Accuracy_score', ascending=False)
```

Model	Accuracy_score
-------	----------------

2	SVM	0.693795
4	Decition Tree	0.690743
3	Random Forest	0.649034
0	KNeighbours	0.643947
1	Naive Bayes	0.632757





## Conclusion

Here SVM classifier has achieved highest accuracy.

[Colab paid products](#) - [Cancel contracts here](#)