WATER POTABILITY PREDICTION

Step-1: Import all the required libraries which are used to train the model or visualise the data

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
import warnings
warnings.filterwarnings("ignore")
import plotly
import plotly.offline as pyo
import plotly.express as px
import plotly.graph_objs as go
pyo.init_notebook_mode()
import plotly.figure_factory as ff
```

Step-2: Read the data

In [2]:
```
df=pd.read_csv("/home/silpa/Downloads/water_potability.csv")
print(df.head())
```

```
         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

   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
```

Step-3: To know number of rows and columns in the data set

In [3]:
```
df.shape
```

Out[3]:
```
(3276, 10)
```

Step-4: Checking for missing values

In [4]:
```
df.isna().sum()
```

Out[4]:
```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

Step-5: To get some information about the data set

In [5]:
```
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
```

Step-6: Describe the dataset which shows the minimum value, maximum value, mean value, count, standard deviation, etc.

In [6]:
```
df.describe()
```

Out[6]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Triha
count	2785.000000	3276.000000	3276.000000	3276.000000	2495.000000	3276.000000	3276.000000	3
mean	7.080795	196.369496	22014.092526	7.122277	333.775777	426.205111	14.284970	
std	1.594320	32.879761	8768.570828	1.583085	41.416840	80.824064	3.308162	
min	0.000000	47.432000	320.942611	0.352000	129.000000	181.483754	2.200000	
25%	6.093092	176.850538	15666.690297	6.127421	307.699498	365.734414	12.065801	
50%	7.036752	196.967627	20927.833607	7.130299	333.073546	421.884968	14.218338	
75%	8.062066	216.667456	27332.762127	8.114887	359.950170	481.792304	16.557652	
max	14.000000	323.124000	61227.196008	13.127000	481.030642	753.342620	28.300000	

Step-7: Filling the missing values using a mean value of each feature.

In [7]:
```
imputer = SimpleImputer(missing_values=np.nan,strategy='mean')
df2 = pd.DataFrame(imputer.fit_transform(df),columns=df.columns)
df2.isnull().sum()
```

Out[7]:
```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
dtype: int64
```

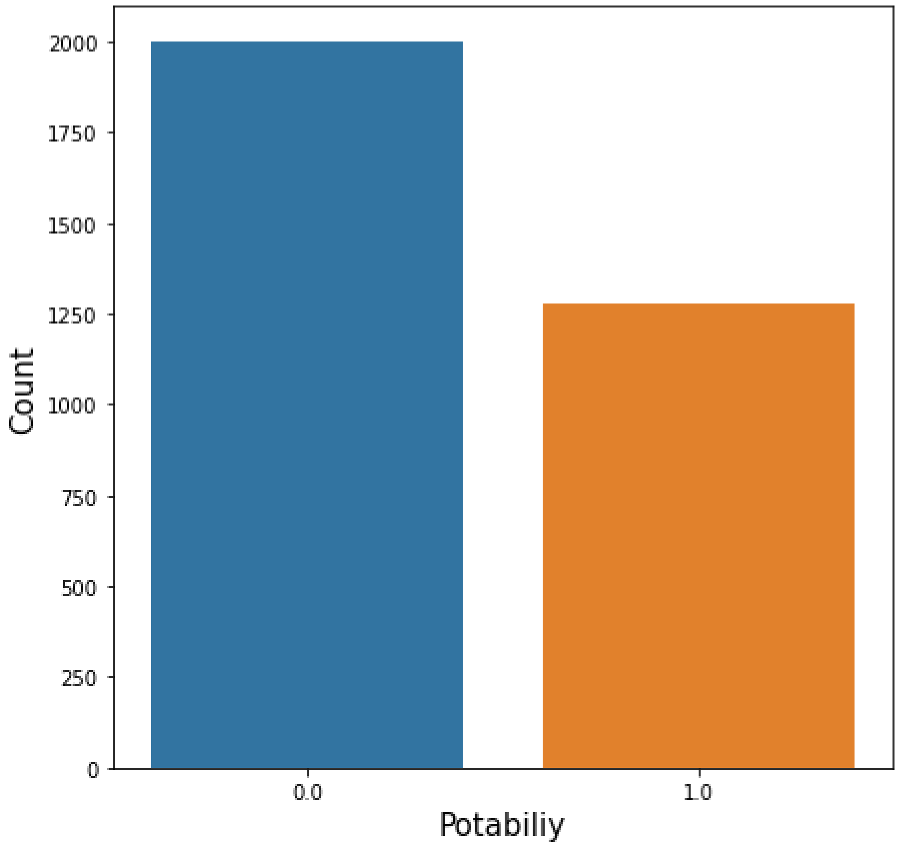
Step-8: Checking the value counts of our target feature Potability.

In [8]:
```
df2.Potability.value_counts()
```

Out[8]:
```
0.0    1998
1.0    1278
Name: Potability, dtype: int64
```

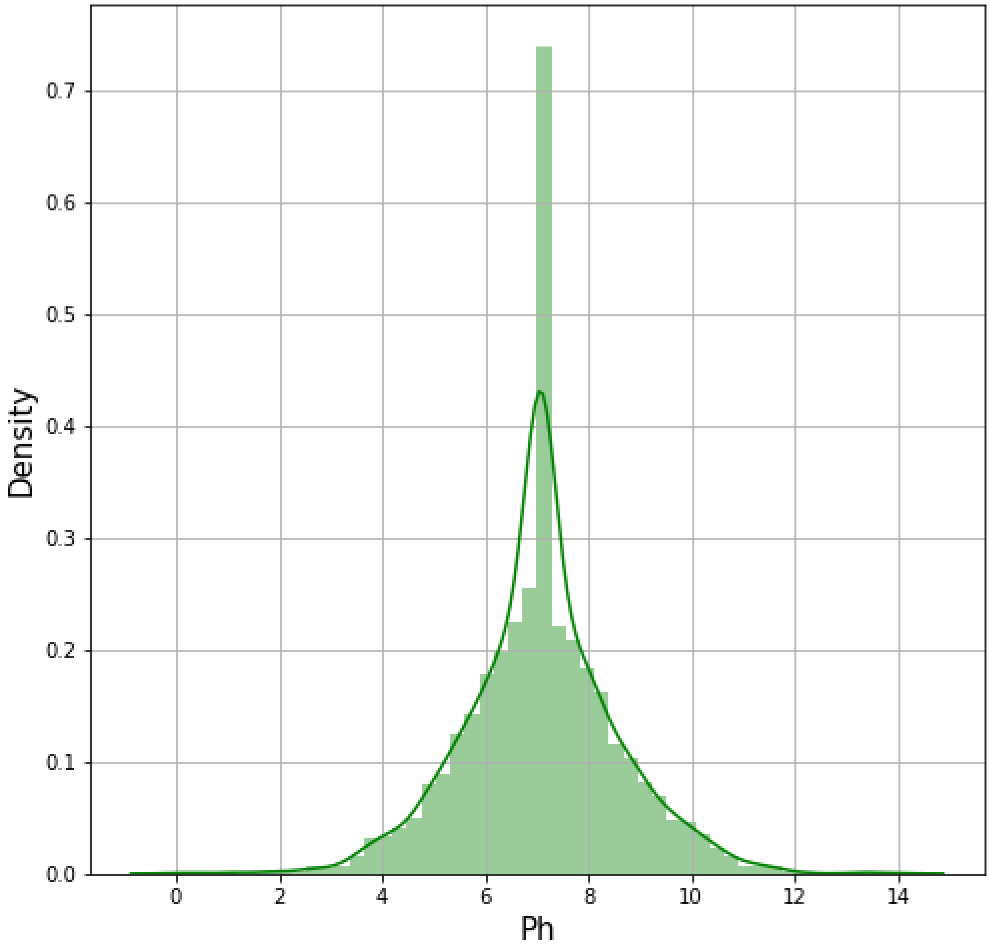
Step-9: Visualising the potability using a countplot function.

In [9]:
```
plt.figure(figsize=(7,7))
sns.countplot(df2['Potability'])
plt.xlabel('Potabiliy',fontsize=15)
plt.ylabel('Count',fontsize=15)
plt.show()
```



Step-10: visualising the pH value using a distplot function.

In [10]:
```
plt.figure(figsize=(8,8))
sns.distplot(df2['ph'],color='green')
plt.xlabel('Ph',fontsize=15)
plt.ylabel('Density',fontsize=15)
plt.grid()
plt.show()
```

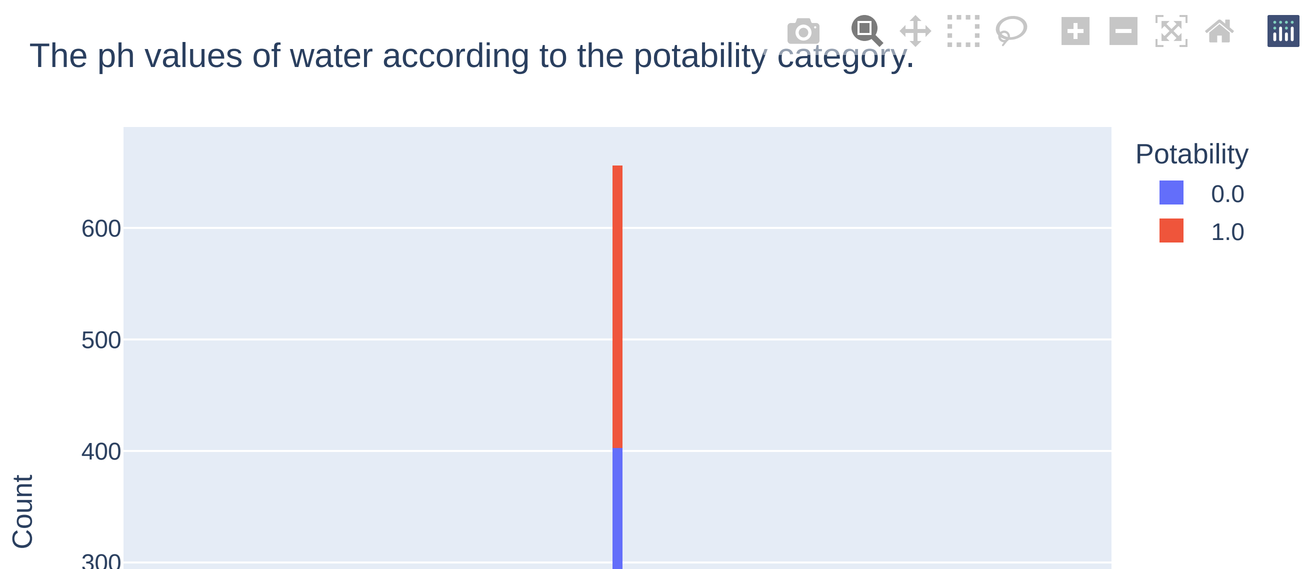


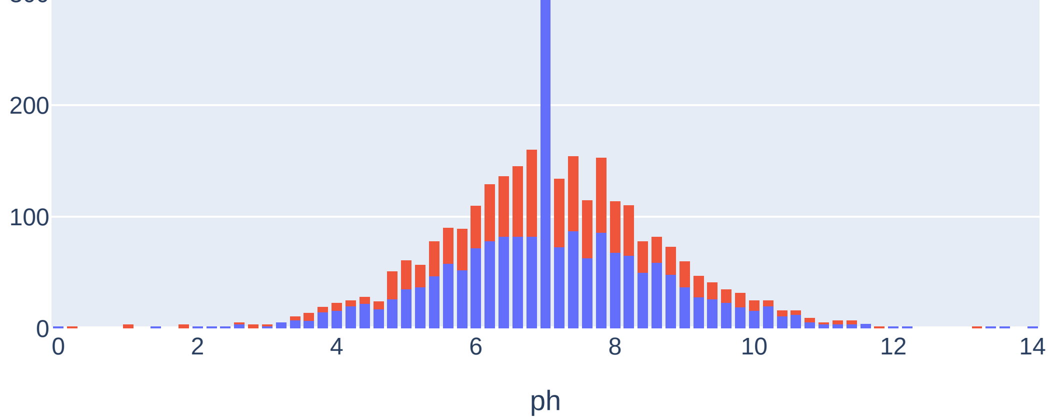
In [11]:
```
df2.columns
```

Out[11]:
```
Index(['ph', 'Hardness', 'Solids', 'Chloramines', 'Sulfate', 'Conductivity',
       'Organic_carbon', 'Trihalomethanes', 'Turbidity', 'Potability'],
      dtype='object')
```

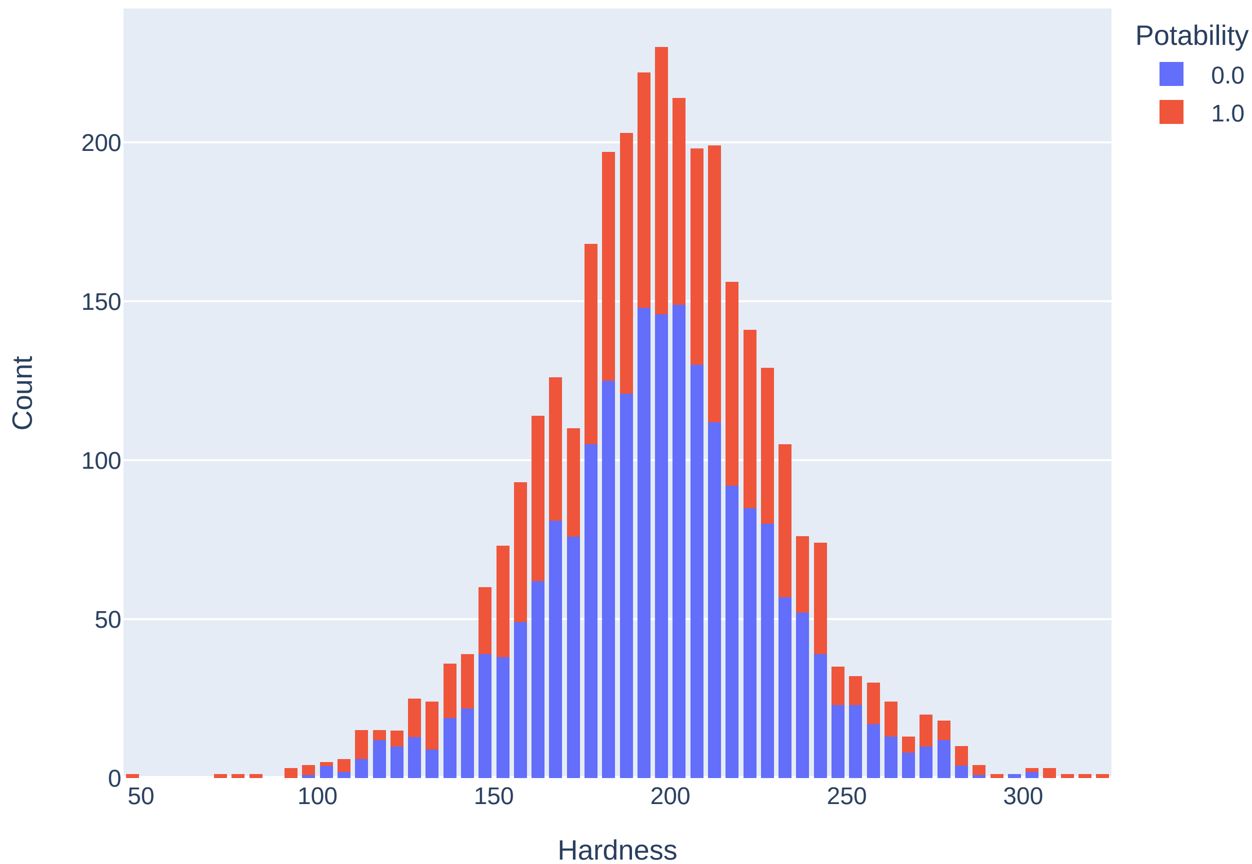
Distribution checking

In [12]:
```
for colm in df2.columns:
    fig = px.histogram(df2, x=colm, color="Potability")
    fig.update_layout(title_text='The ' +colm+ ' values of water according to the potabi
                      xaxis_title_text=colm,
                      yaxis_title_text='Count',
                      bargap=0.2,
                      bargroupgap=0.1)
    fig.show()
```

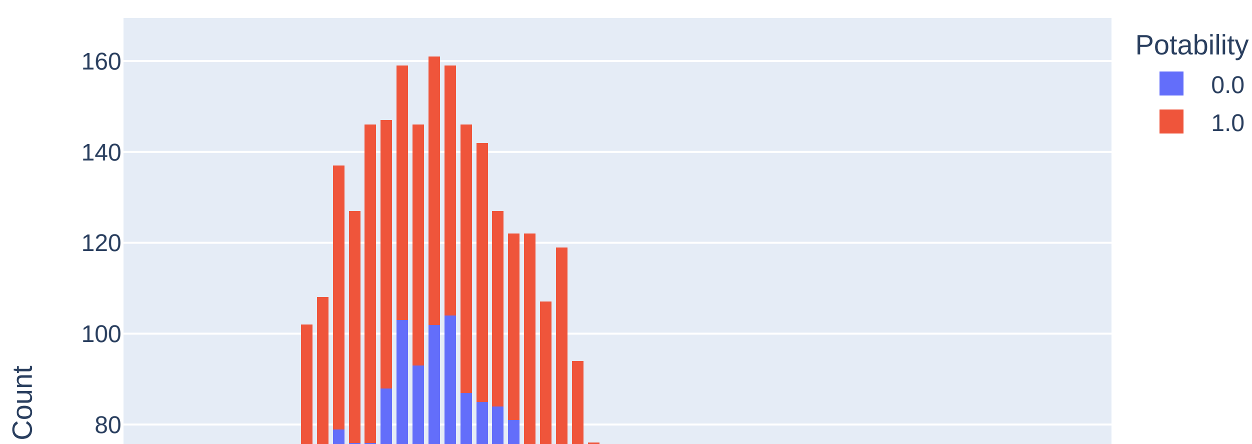


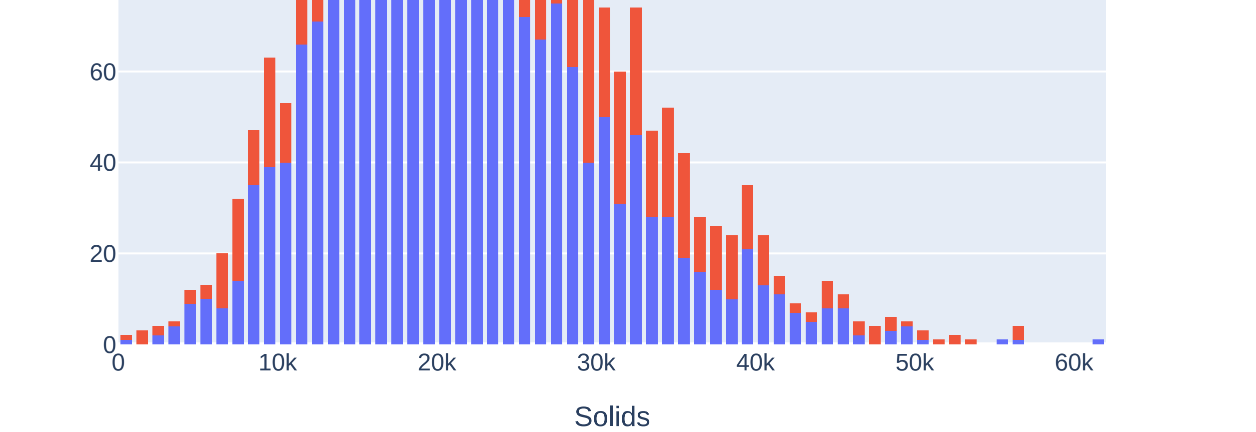


The Hardness values of water according to the potability category.

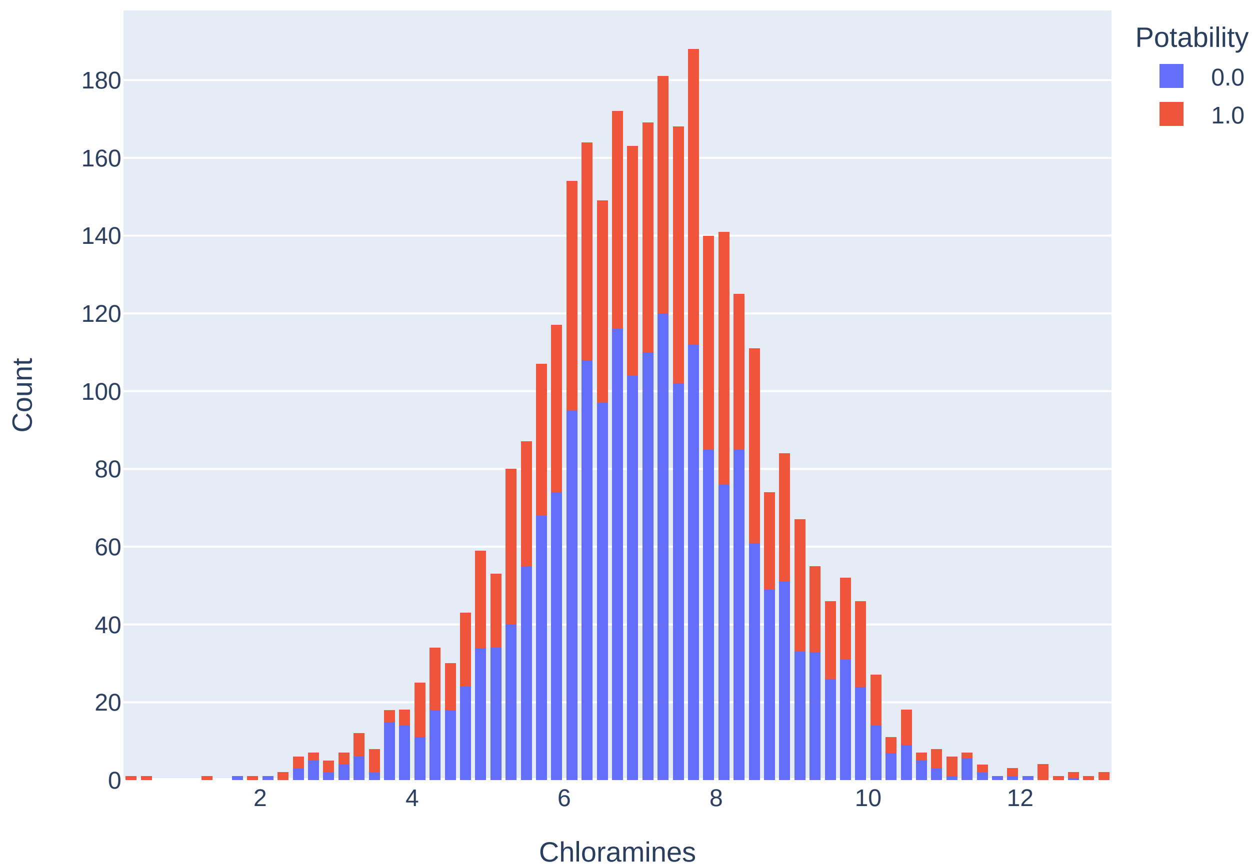


The Solids values of water according to the potability category.

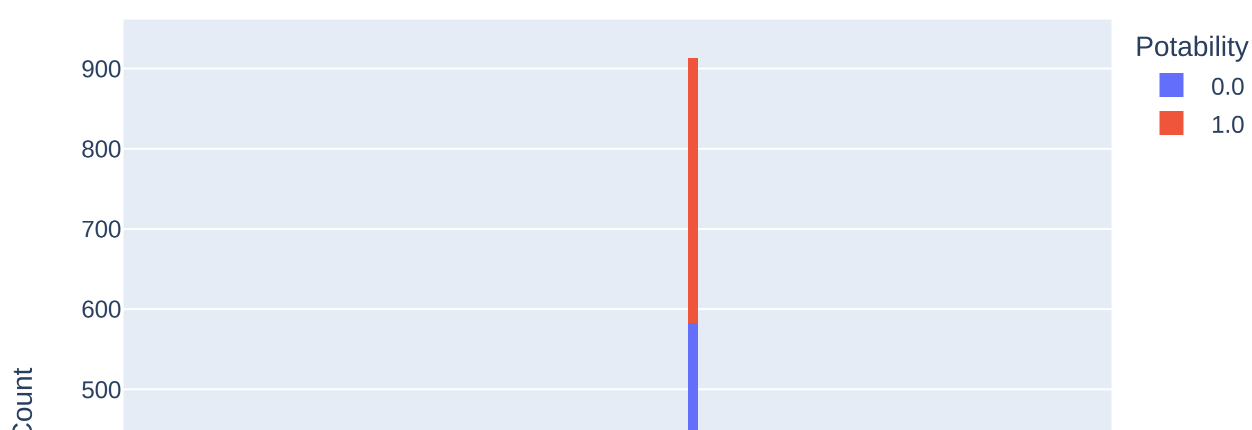


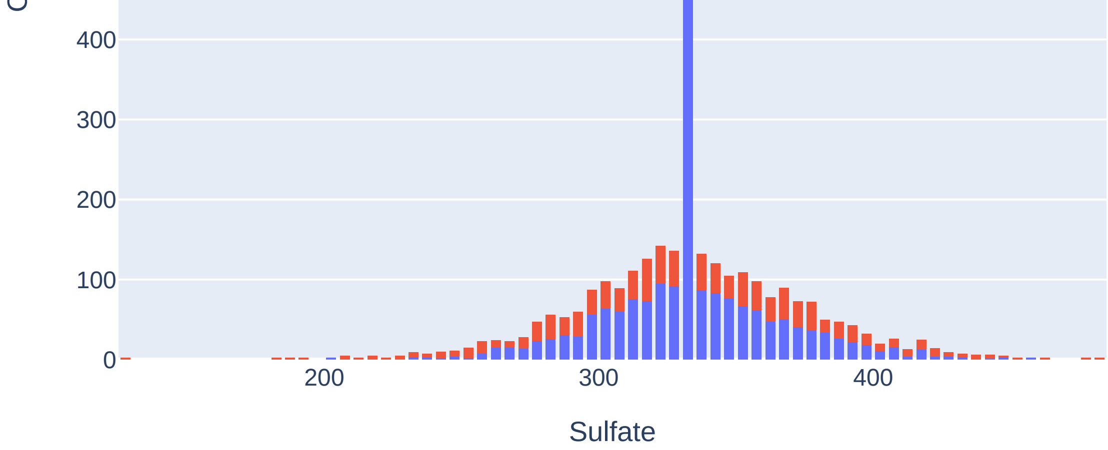


The Chloramines values of water according to the potability category.

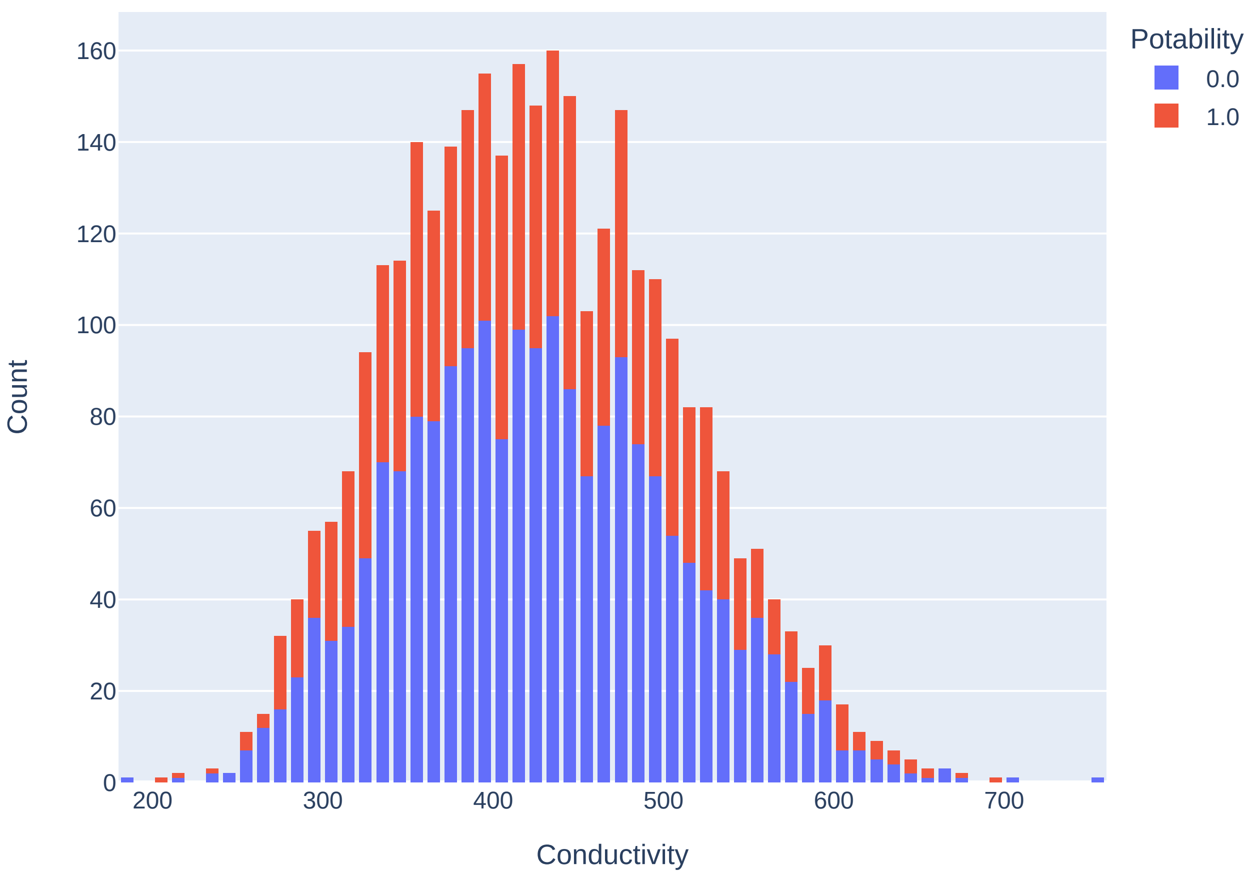


The Sulfate values of water according to the potability category.

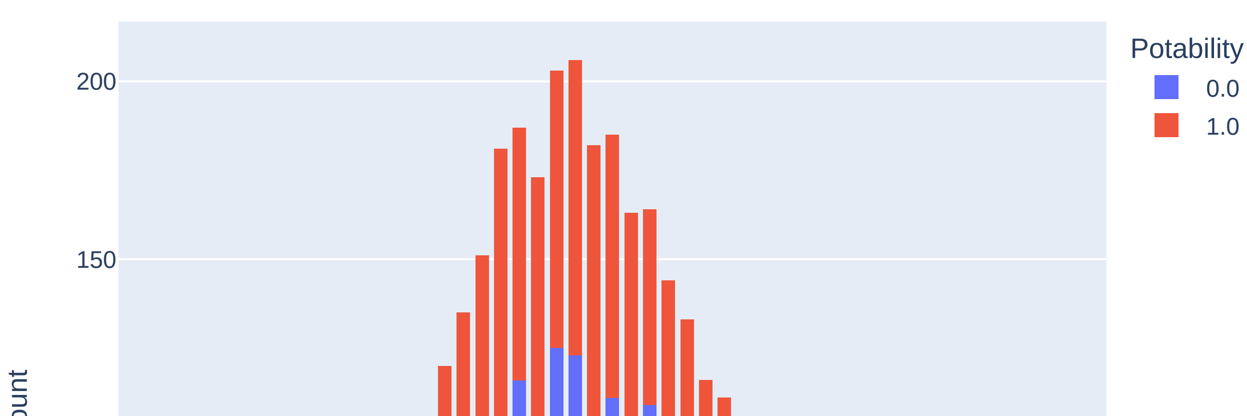


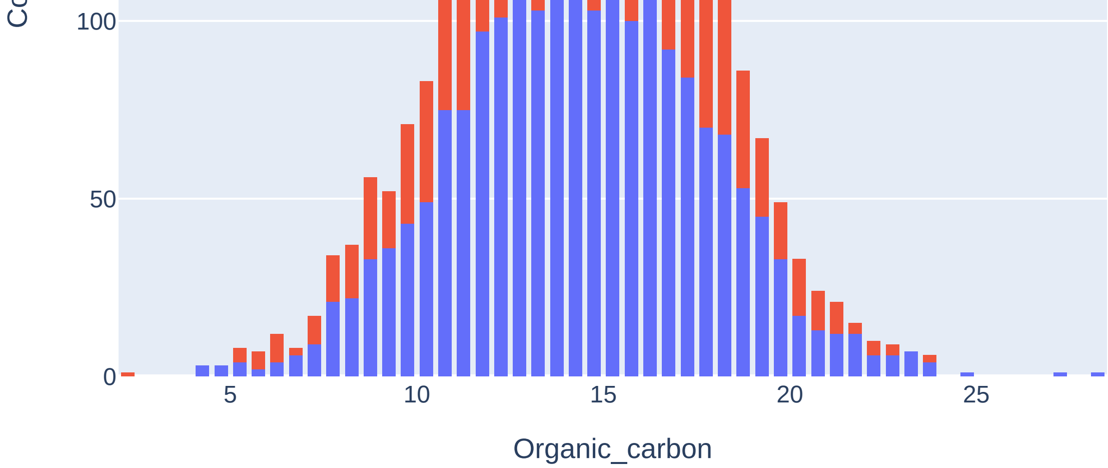


The Conductivity values of water according to the potability category.

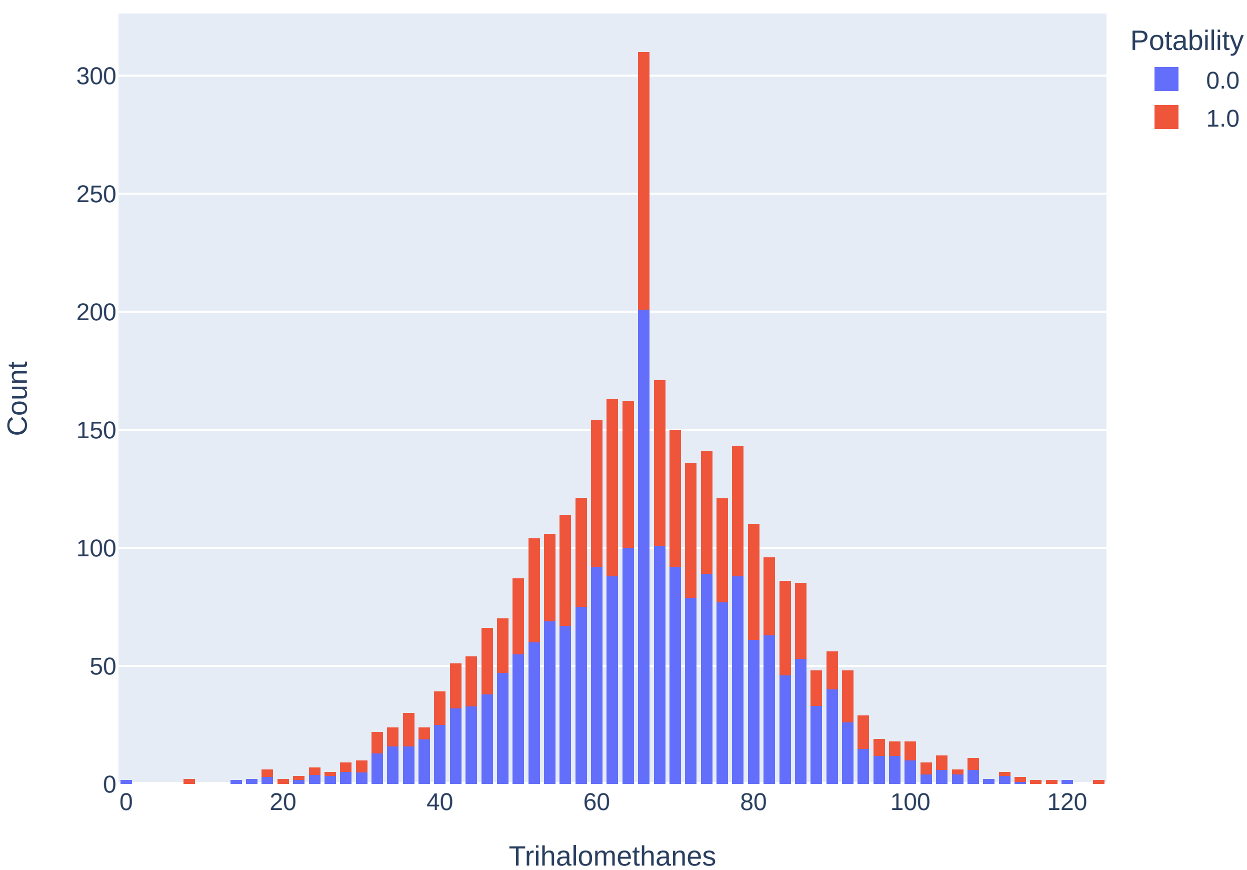


The Organic_carbon values of water according to the potability category.

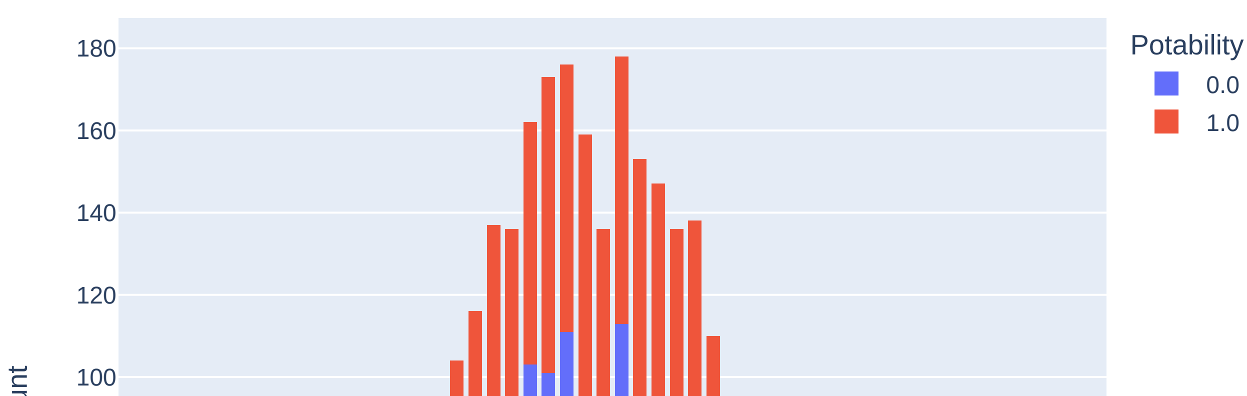


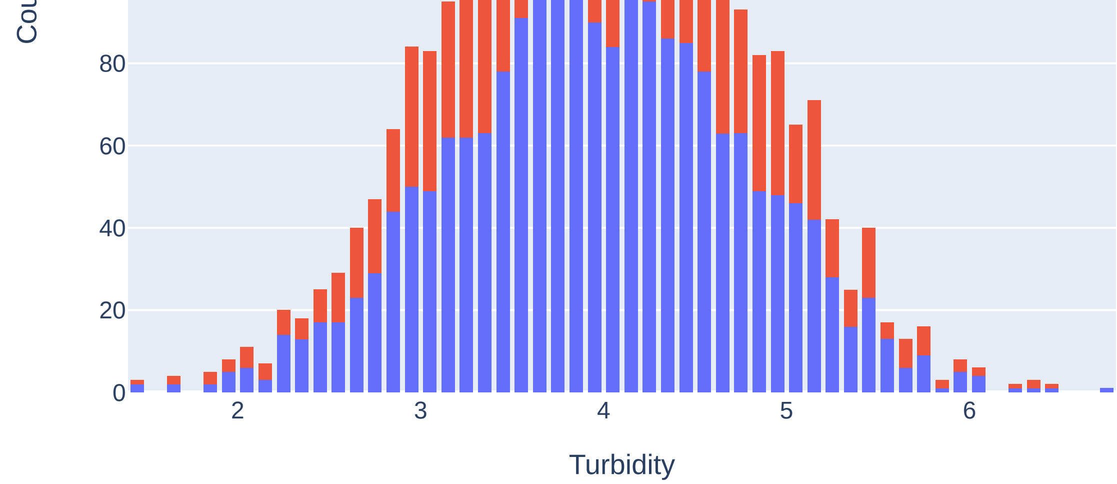


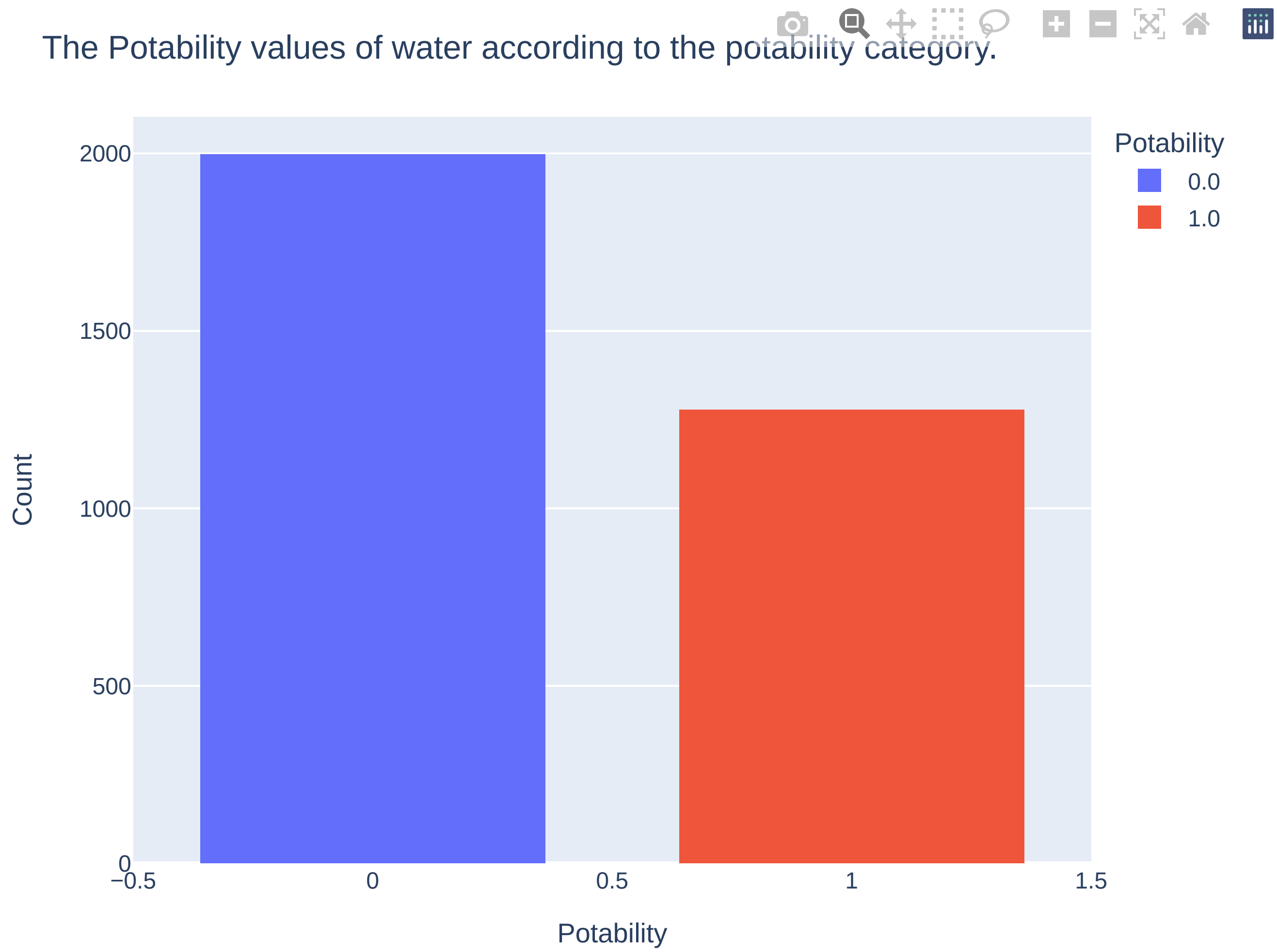
The Trihalomethanes values of water according to the potability category.



The Turbidity values of water according to the potability category.







step-12: Assigning input and output

In [13]:
```
x=df2.iloc[:,:-1].values
print(x)
```

```
[[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]]
```

In [14]:
```
y=df2.iloc[:,-1].values
```

Step-13: Modelling

In [15]:
```
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=101,strat
```

Normalization

In [16]:
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(x_train)
x_train=scaler.transform(x_train)
x_test=scaler.transform(x_test)
```

KNN Algorithm

In [17]:
```
from sklearn.neighbors import KNeighborsClassifier
cl=KNeighborsClassifier(n_neighbors=5)
cl.fit(x_train,y_train)
```

Out[17]:
```
KNeighborsClassifier()
```

In [18]:
```
y_pred=cl.predict(x_test)
print(cl.predict([[7.8,205.56,30700.55,5.2,400.56,66600.2,9.4,89.6,4.6]]))
```

```
[0.]
```

In [19]:
```
from sklearn.metrics import classification_report,confusion_matrix,accuracy_score,plot_c
result=confusion_matrix(y_test,y_pred)
accuracy=accuracy_score(y_test,y_pred)
print(accuracy)
print(result)
r=classification_report(y_test,y_pred)
```

```
0.6408952187182095
[[472 128]
 [225 158]]
```

Naive Bayes Algorithm

In [20]:
```
from sklearn.naive_bayes import GaussianNB
model=GaussianNB()
model.fit(x_train,y_train)
yg_pred=model.predict(x_test)
print(model.predict([[7.8,205.56,30700.55,5.2,400.56,66600.2,9.4,89.6,4.6]]))
```

```
[1.]
```

In [21]:
```
cn=confusion_matrix(y_test,yg_pred)
print(cn)
acc=accuracy_score(y_test,yg_pred)
print(acc)
rpt=classification_report(y_test,yg_pred)
```

```
[[534  66]
 [302  81]]
```

0.6256358087487284

Support Vector Machine Algorithm

In [22]:
```
from sklearn.svm import SVC
classifier=SVC()
classifier.fit(x_train,y_train)
ys_pred=classifier.predict(x_test)
print(classifier.predict([[7.8,205.56,30700.55,5.2,400.56,66600.2,9.4,89.6,4.6]]))
```

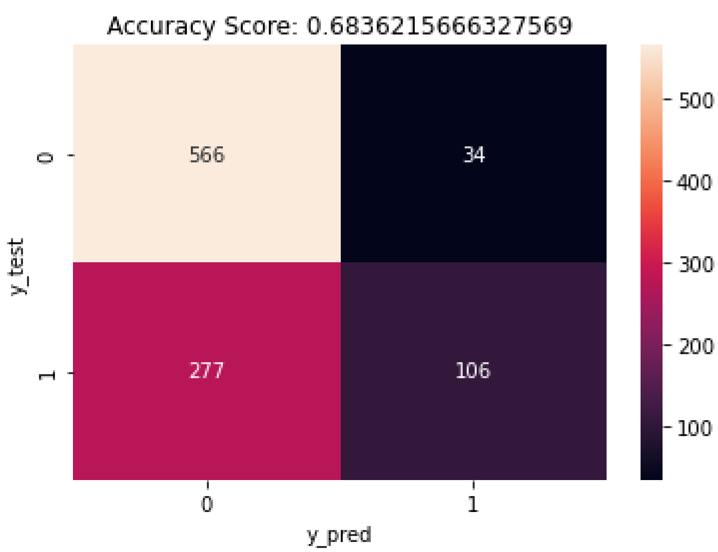
[1.]

In [23]:
```
cnf=confusion_matrix(y_test,ys_pred)
print(cnf)
ac=accuracy_score(y_test,ys_pred)
print(ac)
rprt=classification_report(y_test,ys_pred)
```

```
[[566  34]
 [277 106]]
0.6836215666327569
```

Comparison

In [24]:
```
cm=confusion_matrix(y_test,ys_pred)
sns.heatmap(cm,annot=True,fmt=".0f")
plt.xlabel('y_pred')
plt.ylabel('y_test')
plt.title('Accuracy Score: {0}'.format(ac), size=12)
plt.show()
print('KNN:',r,"*" * 100,"\n","\n",'NAIVE_BAYES:',rpt,"*" * 100,"\n","\n",'SVC:',rprt)
```




```
KNN:               precision    recall  f1-score   support

         0.0       0.68      0.79      0.73       600
         1.0       0.55      0.41      0.47       383

    accuracy                           0.64       983
   macro avg       0.61      0.60      0.60       983
weighted avg       0.63      0.64      0.63       983
 ****************************************************************************************
*************

 NAIVE_BAYES:               precision    recall  f1-score   support

         0.0       0.64      0.89      0.74       600
         1.0       0.55      0.21      0.31       383
```

	precision	recall	f1-score	support
accuracy			0.63	983
macro avg	0.59	0.55	0.52	983
weighted avg	0.60	0.63	0.57	983

****

SVC:	precision	recall	f1-score	support
0.0	0.67	0.94	0.78	600
1.0	0.76	0.28	0.41	383
accuracy			0.68	983
macro avg	0.71	0.61	0.59	983
weighted avg	0.70	0.68	0.64	983