In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries inst
# It is defined by the kaggle/python Docker image: https://github.com/kaggle
# For example, here's several helpful packages to load
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from tqdm import tqdm_notebook
import plotly.figure_factory as ff

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

import warnings
warnings.filterwarnings('ignore')

plt.style.use('fivethirtyeight')
%matplotlib inline
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will l

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that
# You can also write temporary files to /kaggle/temp/, but they won't be sav
```

```
/kaggle/input/water-potability/water_potability.csv
```

In [2]:

```
data=pd.read_csv('../input/water-potability/water_potability.csv')
data.head()
```

Out[2]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_c
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.3
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.1
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.8
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.4
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.5

EDA

- ph-> pH of water
- Hardness-> Capacity of water to precipitate soap in mg/L
- Solids-> Total dissolved solids in ppm
- Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js ines 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 emiting property of water in NTU (Nephelometric Turbidity Units)
- Potability-> Indicates if water is safe for human consumption

In [3]:
```
data.describe()
```

Out[3]:

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

In [4]:
```
data.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
```

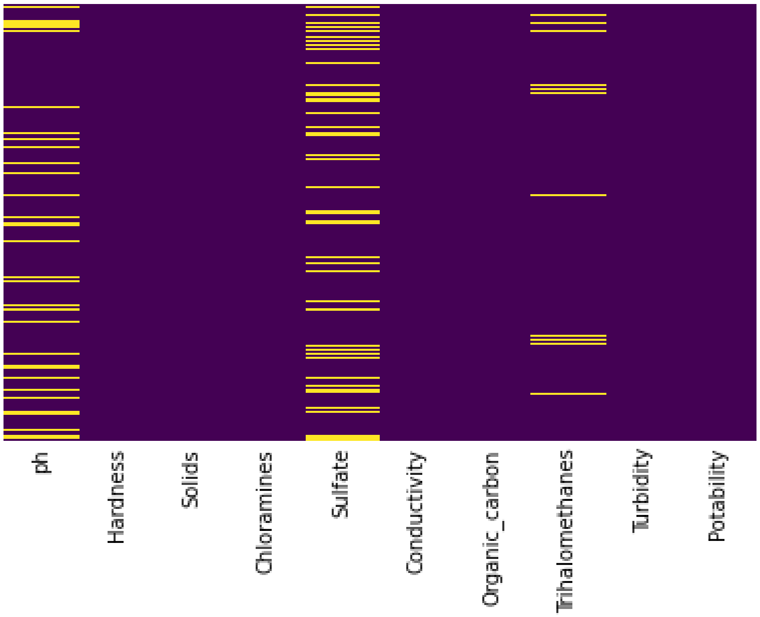
In [5]:
```
print('There are {} data points and {} features in the data'.format(data.sha
```

```
There are 3276 data points and 10 features in the data
```

Null Values

```
In [6]: sns.heatmap(data.isnull(),yticklabels=False,cbar=False,cmap='viridis')
```

```
Out[6]: <AxesSubplot:>
```



```
In [7]: for i in data.columns:
            if data[i].isnull().sum()>0:
                print("There are {} null values in {} column".format(data[i].isnull(
```

```
There are 491 null values in ph column
There are 781 null values in Sulfate column
There are 162 null values in Trihalomethanes column
```

Handling Null Values

PH

```
In [8]: data['ph'].describe()
```

```
Out[8]: count    2785.000000
        mean        7.080795
        std         1.594320
        min         0.000000
        25%         6.093092
        50%         7.036752
        75%         8.062066
        max        14.000000
        Name: ph, dtype: float64
```

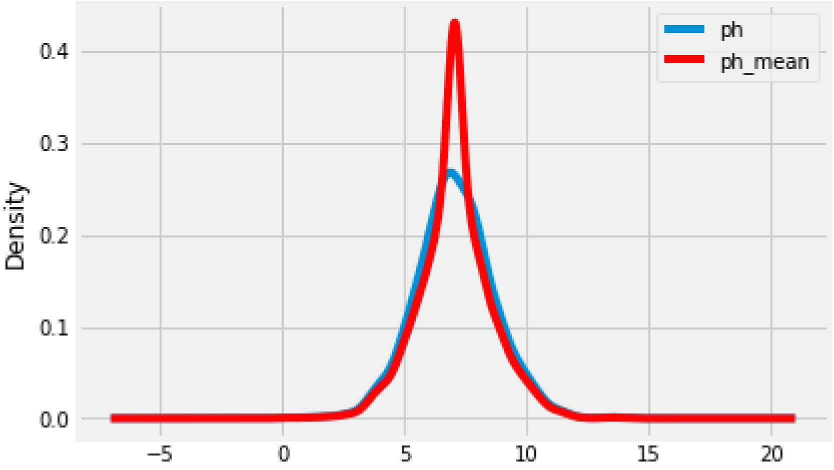
Filling the missing values by mean

```
In [9]: data['ph_mean']=data['ph'].fillna(data['ph'].mean())
```

```
In [10]: data['ph_mean'].isnull().sum()
```

Out[10]: 0

In [11]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
data['ph'].plot(kind='kde', ax=ax)
data.ph_mean.plot(kind='kde', ax=ax, color='red')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
plt.show()
```



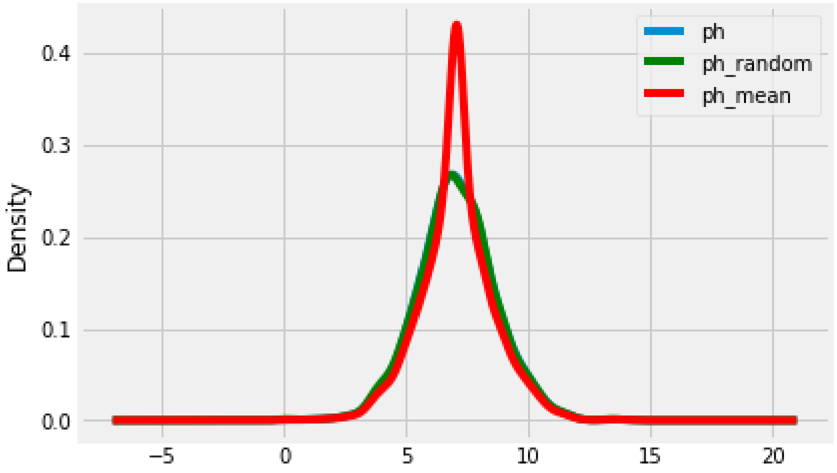
The distribution is not uniform

Filling the data with random values

In [12]:
```
def impute_nan(df,variable):
    df[variable+"_random"]=df[variable]
    ##It will have the random sample to fill the na
    random_sample=df[variable].dropna().sample(df[variable].isnull().sum(),r
    ##pandas need to have same index in order to merge the dataset
    random_sample.index=df[df[variable].isnull()].index
    df.loc[df[variable].isnull(),variable+'_random']=random_sample
```

In [13]:
```
impute_nan(data,"ph")
```

In [14]:
```
fig = plt.figure()
ax = fig.add_subplot(111)
data['ph'].plot(kind='kde', ax=ax)
data.ph_random.plot(kind='kde', ax=ax, color='green')
data.ph_mean.plot(kind='kde', ax=ax, color='red')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
plt.show()
```



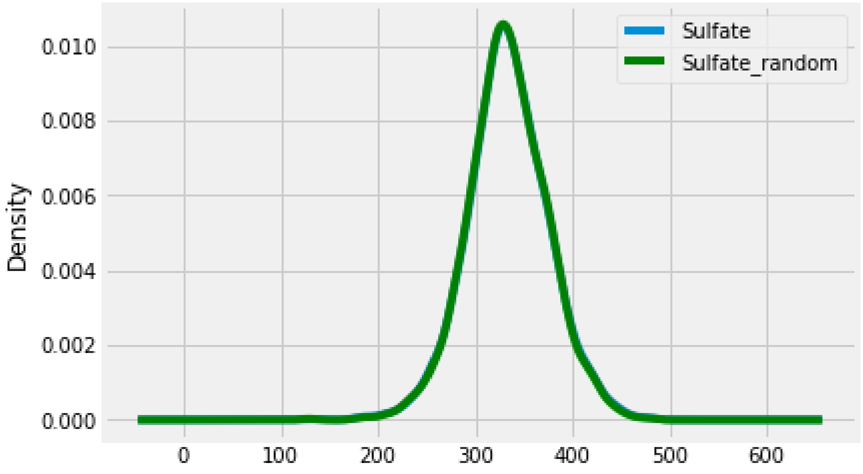
Uniform distribution with random initialization

In [15]:

```
impute_nan(data,"Sulfate")
```

In [16]:

```
fig = plt.figure()
ax = fig.add_subplot(111)
data['Sulfate'].plot(kind='kde', ax=ax)
data["Sulfate_random"].plot(kind='kde', ax=ax, color='green')
#data.ph_mean.plot(kind='kde', ax=ax, color='red')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
plt.show()
```

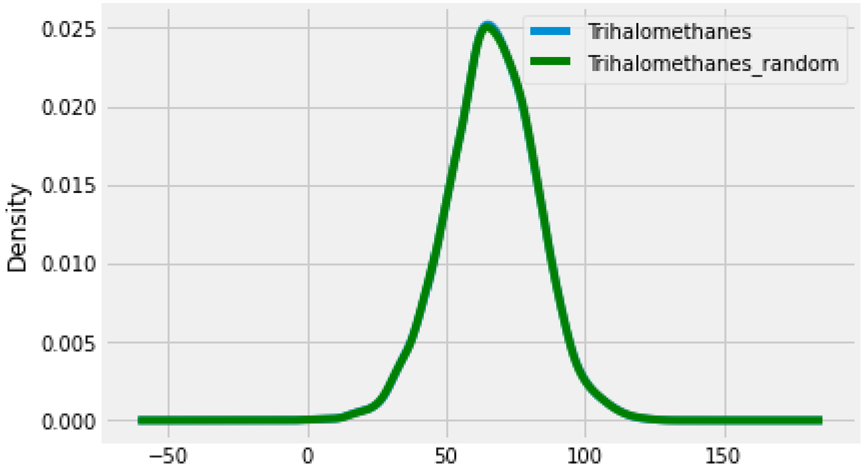


In [17]:

```
impute_nan(data,"Trihalomethanes")
```

In [18]:

```
fig = plt.figure()
ax = fig.add_subplot(111)
data['Trihalomethanes'].plot(kind='kde', ax=ax)
data.Trihalomethanes_random.plot(kind='kde', ax=ax, color='green')
lines, labels = ax.get_legend_handles_labels()
ax.legend(lines, labels, loc='best')
plt.show()
```



In [19]:
```
data=data.drop(['ph','Sulfate','Trihalomethanes','ph_mean'],axis=1)
```

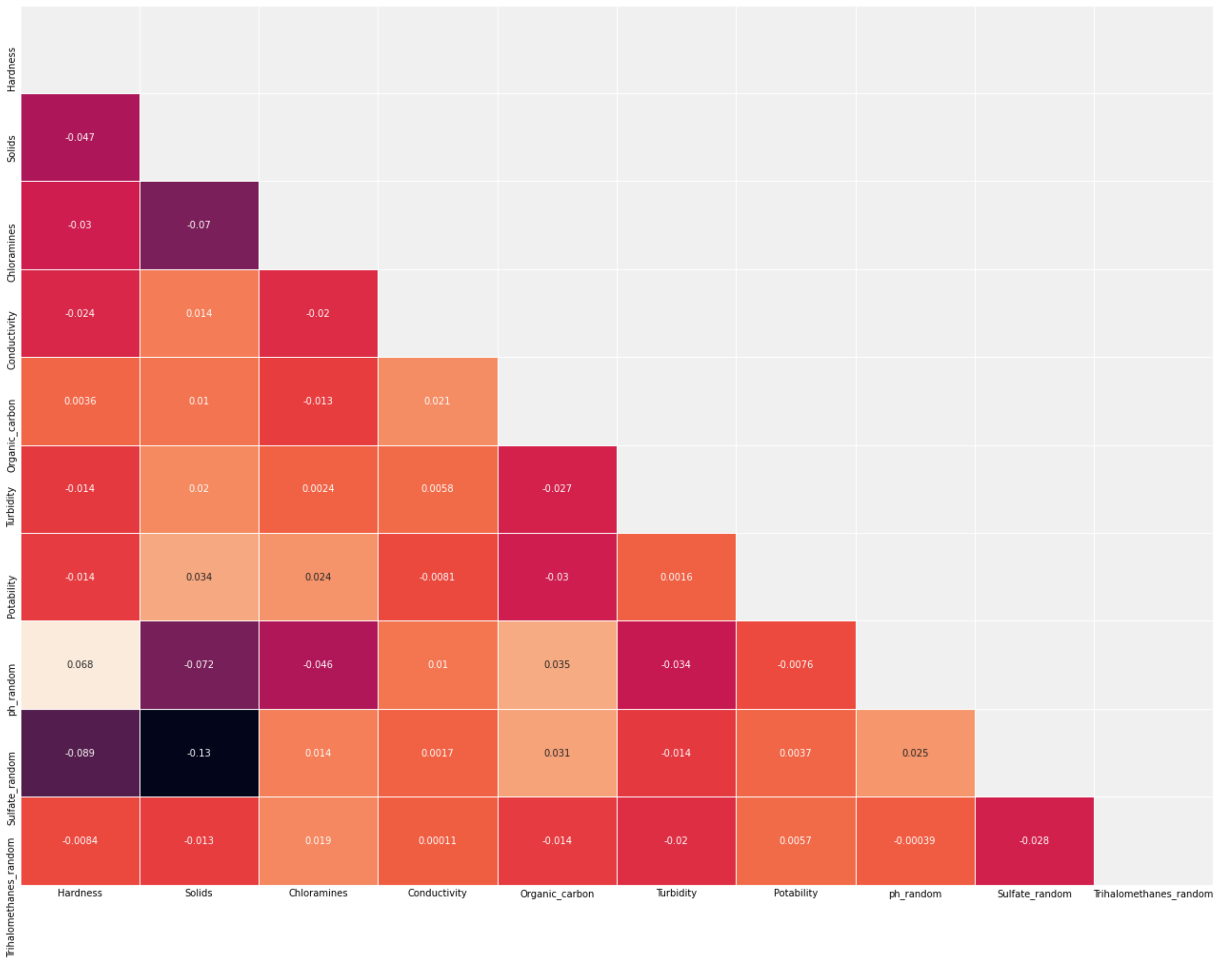
In [20]:
```
data.isnull().sum()
```

Out[20]:
```
Hardness                  0
Solids                    0
Chloramines               0
Conductivity              0
Organic_carbon            0
Turbidity                 0
Potability                0
ph_random                 0
Sulfate_random            0
Trihalomethanes_random    0
dtype: int64
```

Check for Correlation

In [21]:
```
plt.figure(figsize=(20, 17))
matrix = np.triu(data.corr())
sns.heatmap(data.corr(), annot=True,linewidth=.8, mask=matrix, cmap="rocket"
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

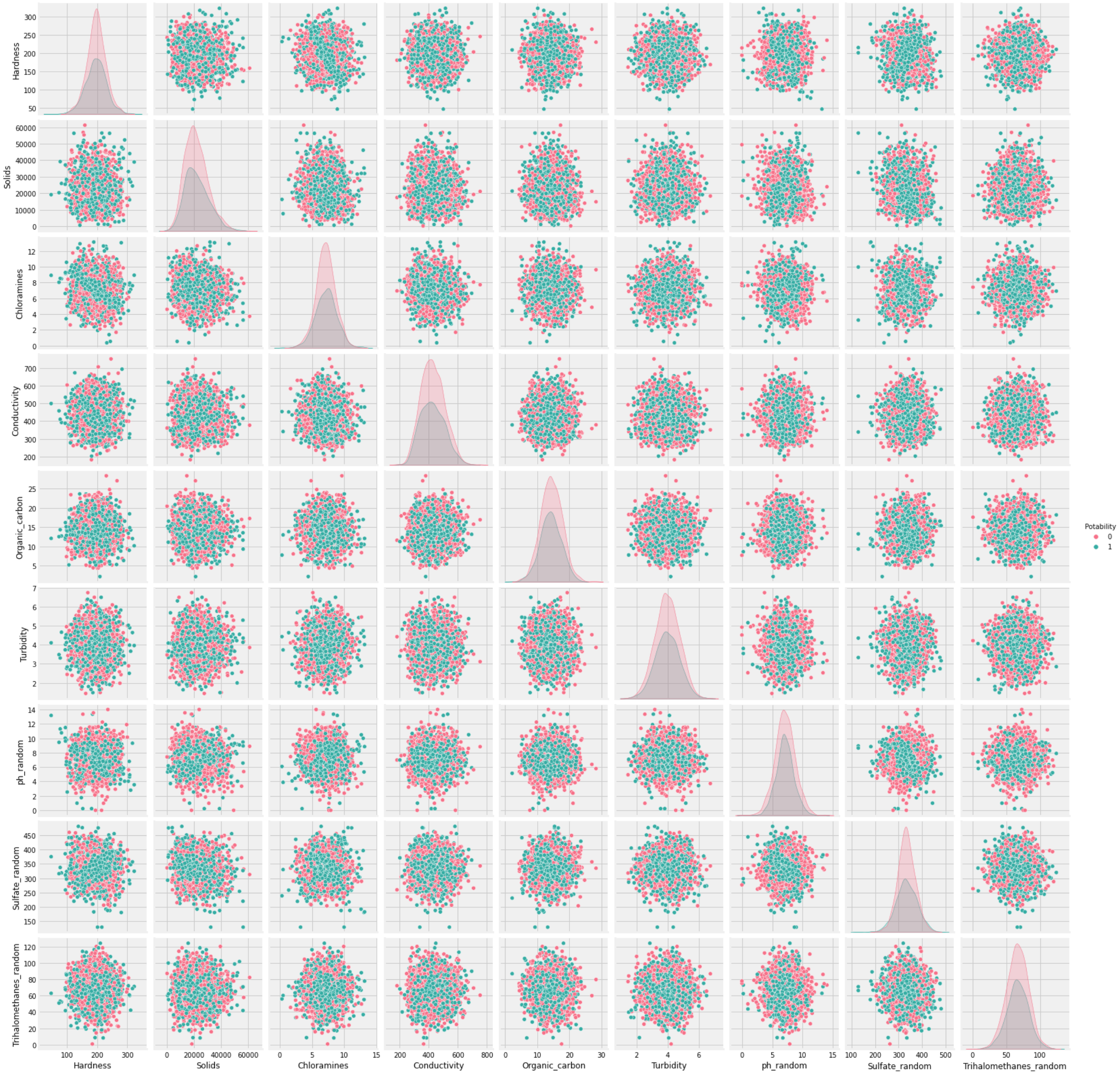


There are no correlated columns presebt in the data

In [22]:

```
sns.pairplot(data, hue="Potability", palette="husl");
```

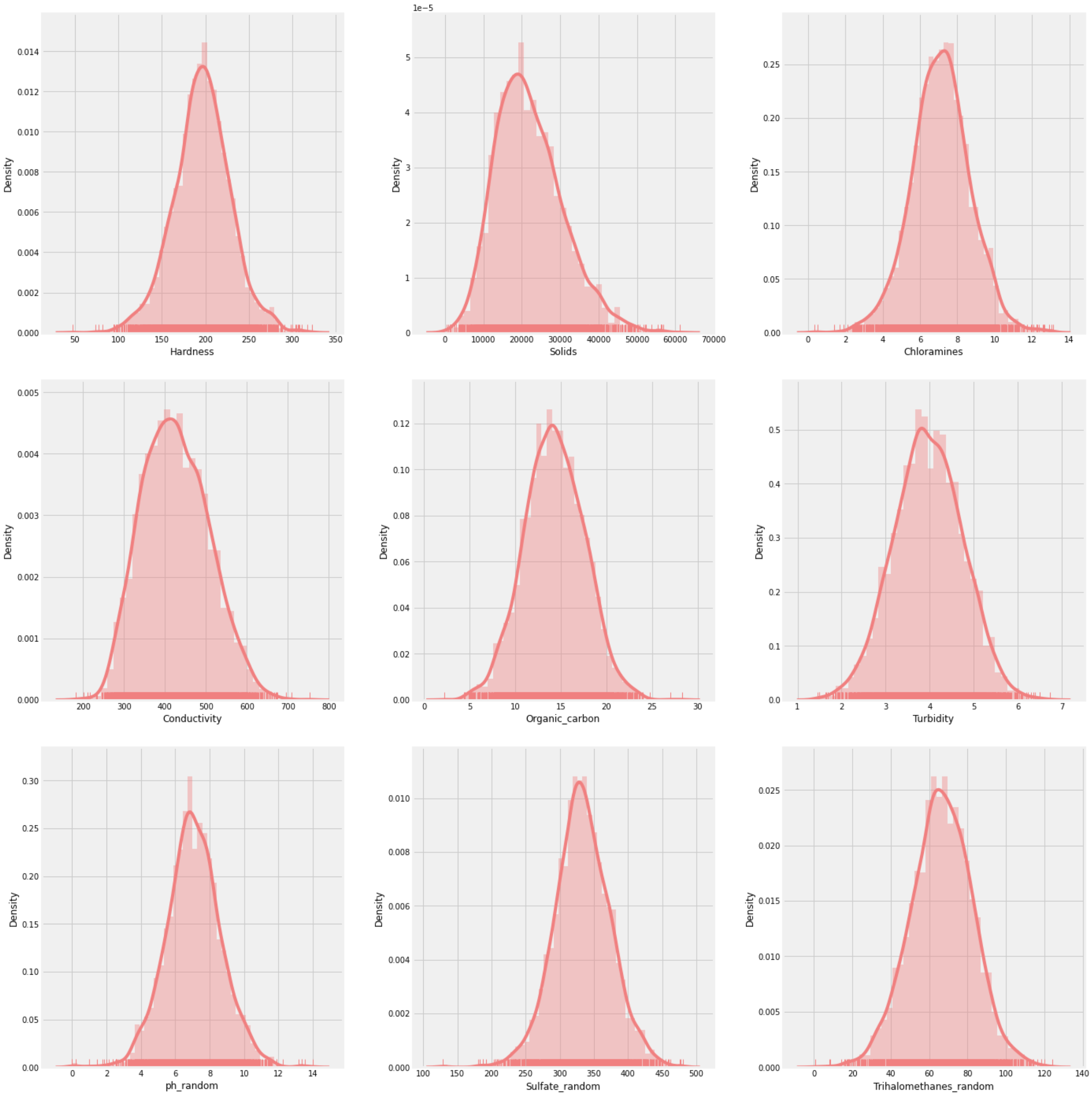
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [23]:

```python
def distributionPlot(data):
    """
    Creates distribution plot.
    """
    fig = plt.figure(figsize=(20, 20))
    for i in tqdm_notebook(range(0, len(data.columns))):
        fig.add_subplot(np.ceil(len(data.columns)/3), 3, i+1)
        sns.distplot(
            data.iloc[:, i], color="lightcoral", rug=True)
        fig.tight_layout(pad=3.0)
plot_data = data.drop(['Potability'], axis =1)
distributionPlot(plot_data)
```

```
0%|          | 0/9 [00:00<?, ?it/s]
```



Hardness

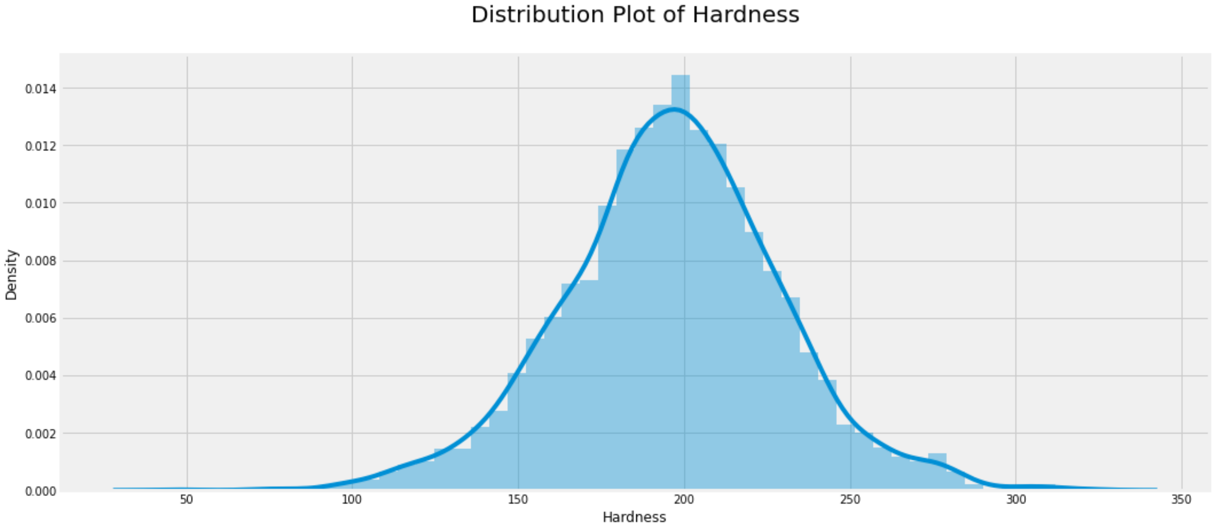
```
In [24]: data['Hardness'].describe()
```

```
Out[24]: count    3276.000000
         mean      196.369496
         std        32.879761
         min        47.432000
         25%       176.850538
         50%       196.967627
         75%       216.667456
         max       323.124000
         Name: Hardness, dtype: float64
```

```
In [25]: plt.figure(figsize = (16, 7))
         sns.distplot(data['Hardness'])
```

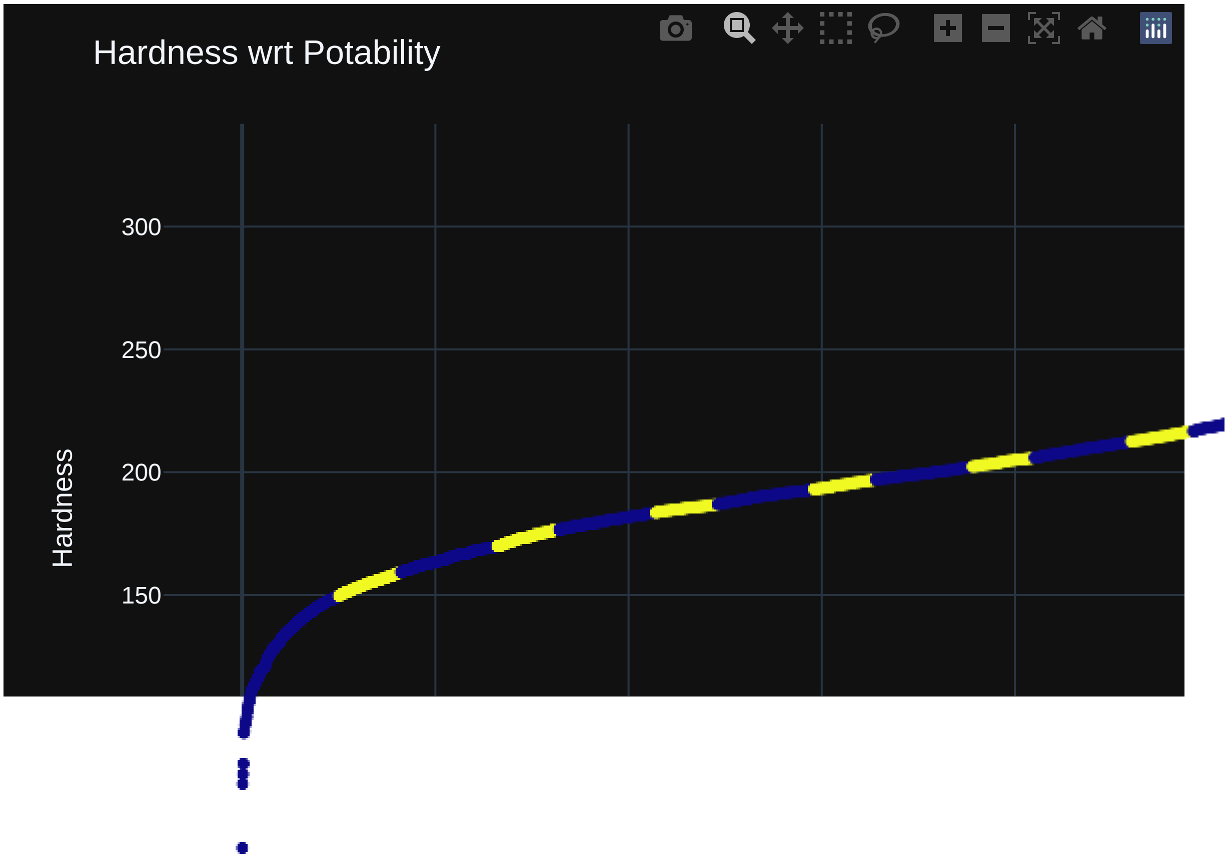
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
plt.title('Distribution Plot of Hardness\n', fontsize =  20)
plt.show()
```



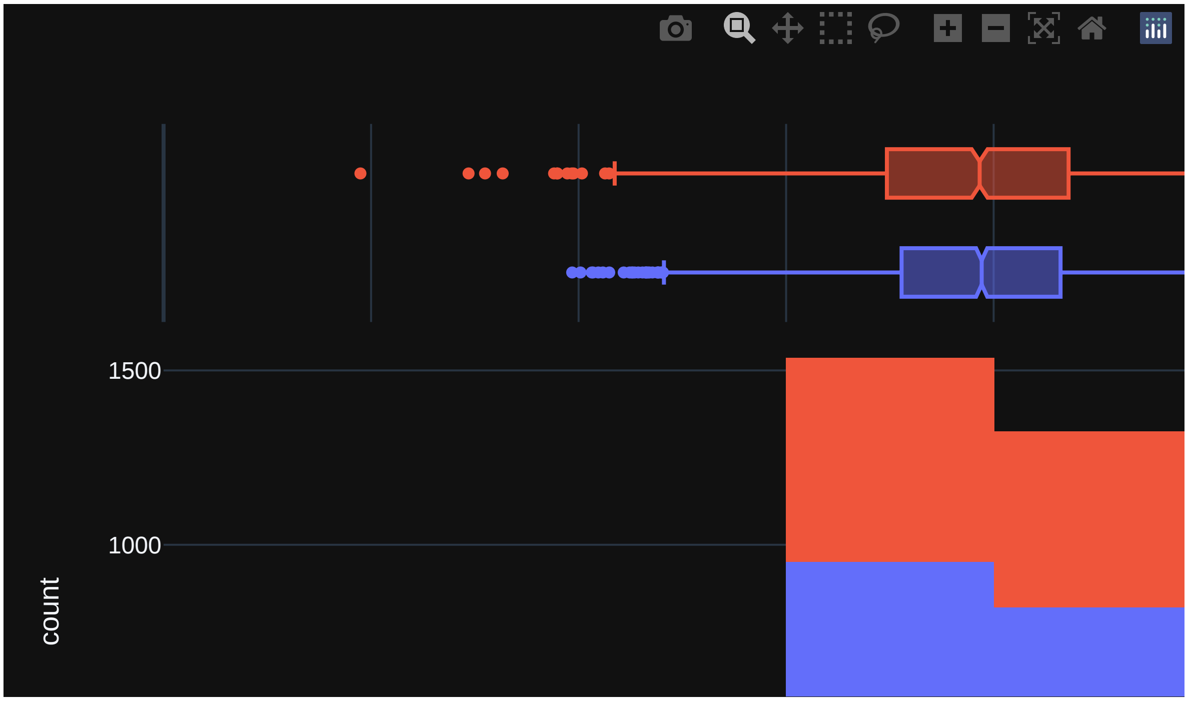
In [26]:

```
# basic scatter plot
fig = px.scatter(data,range(data['Hardness'].count()), sorted(data['Hardness
                 color=data['Potability'],
                 labels={
                     'x': "Count",
                     'y': "Hardness",
                     'color':'Potability'

                 }, template = 'plotly_dark')
fig.update_layout(title='Hardness wrt Potability')
fig.show()
```



```
In [27]: px.histogram(data_frame = data, x = 'Hardness', nbins = 10, color = 'Potabil
                      template = 'plotly_dark')
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

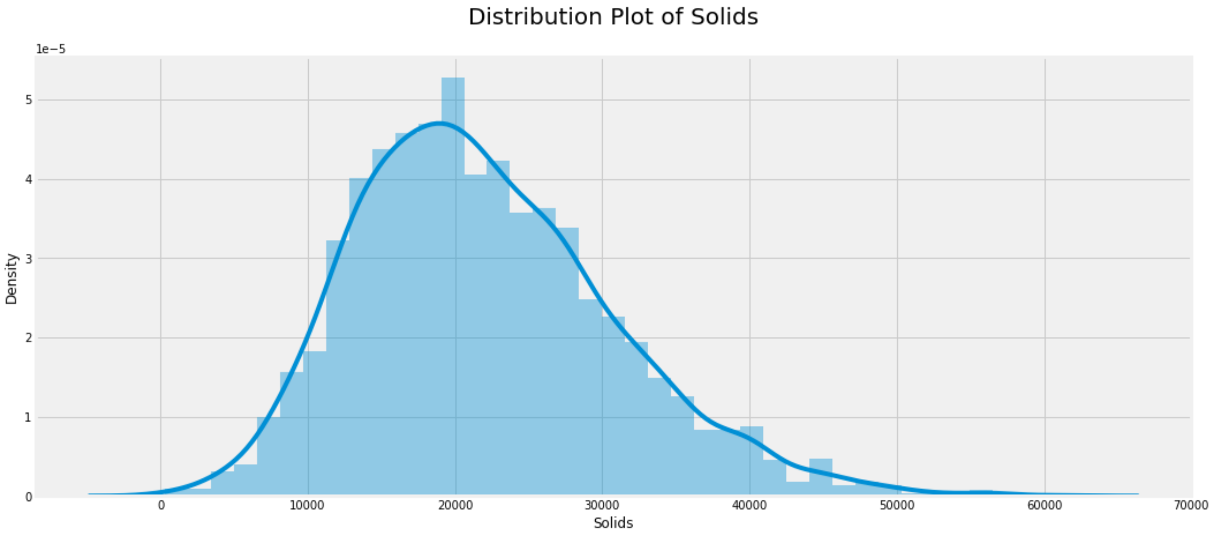


Solids

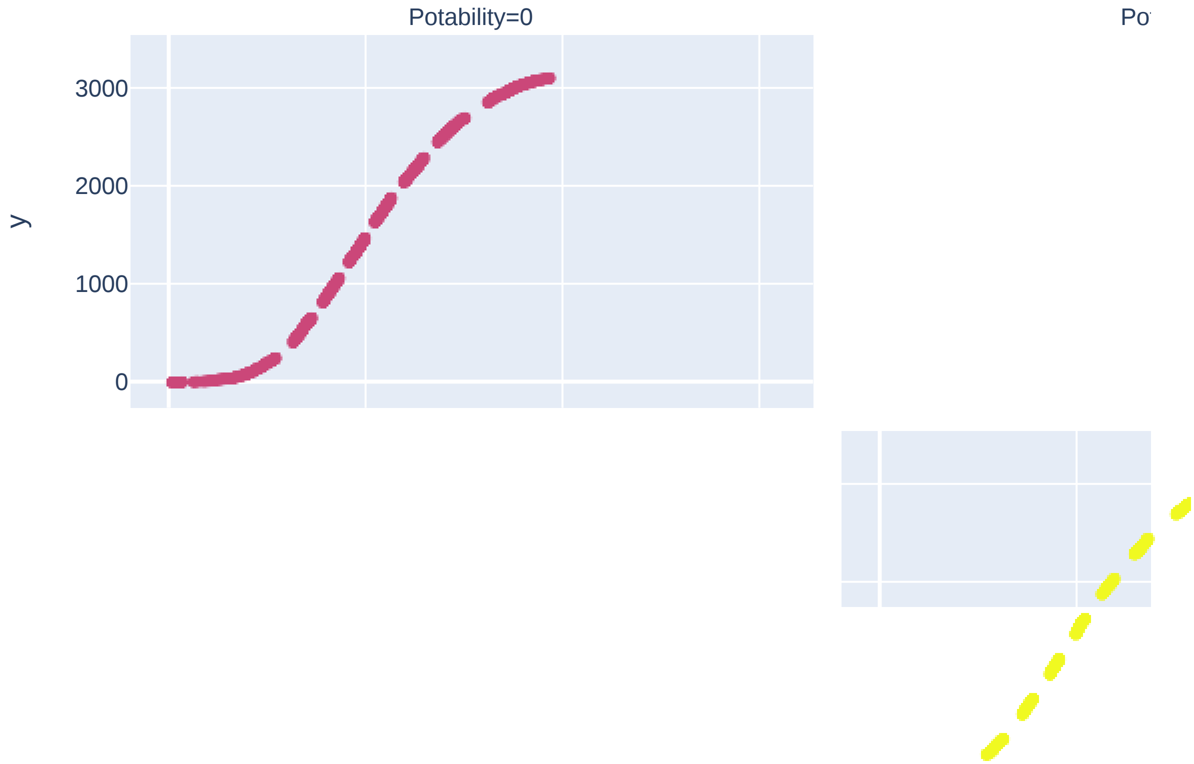
```
In [28]: data['Solids'].describe()
```

```
Out[28]: count     3276.000000
         mean     22014.092526
         std       8768.570828
         min        320.942611
         25%      15666.690297
         50%      20927.833607
         75%      27332.762127
         max      61227.196008
         Name: Solids, dtype: float64
```

```
In [29]: plt.figure(figsize = (16, 7))
         sns.distplot(data['Solids'])
         plt.title('Distribution Plot of Solids\n', fontsize =  20)
         plt.show()
```

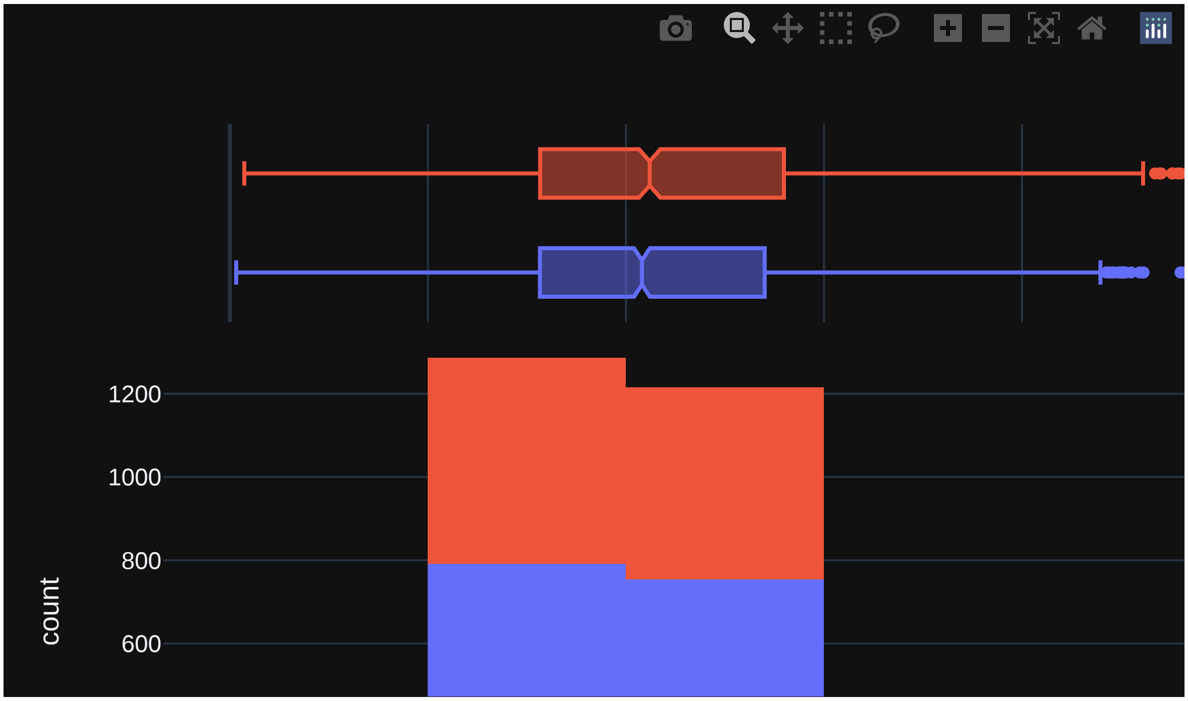


In [30]:
```
fig = px.scatter(data, sorted(data["Solids"]), range(data["Solids"].count())
                 facet_row="Potability")
fig.show()
```



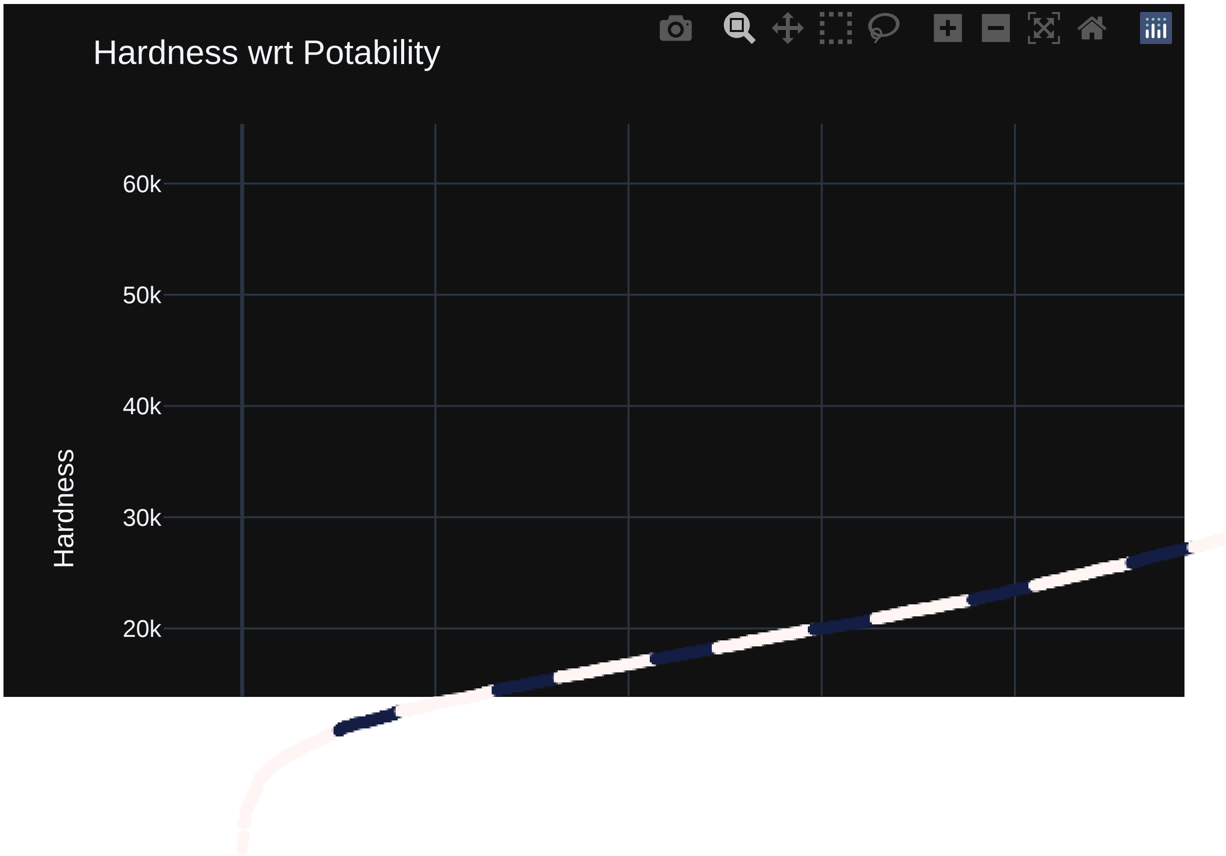
In [31]:
```
px.histogram(data_frame = data, x = 'Solids', nbins = 10, color = 'Potabilit
             template = 'plotly_dark')
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [32]:

```
# basic scatter plot
fig = px.scatter(data,range(data['Solids'].count()), sorted(data['Solids']),
                 color=data['Potability'],
                 labels={
                     'x': "Count",
                     'y': "Hardness",
                     'color':'Potability'

                 },
                 color_continuous_scale=px.colors.sequential.tempo,
                 template = 'plotly_dark')
fig.update_layout(title='Hardness wrt Potability')
fig.show()
```



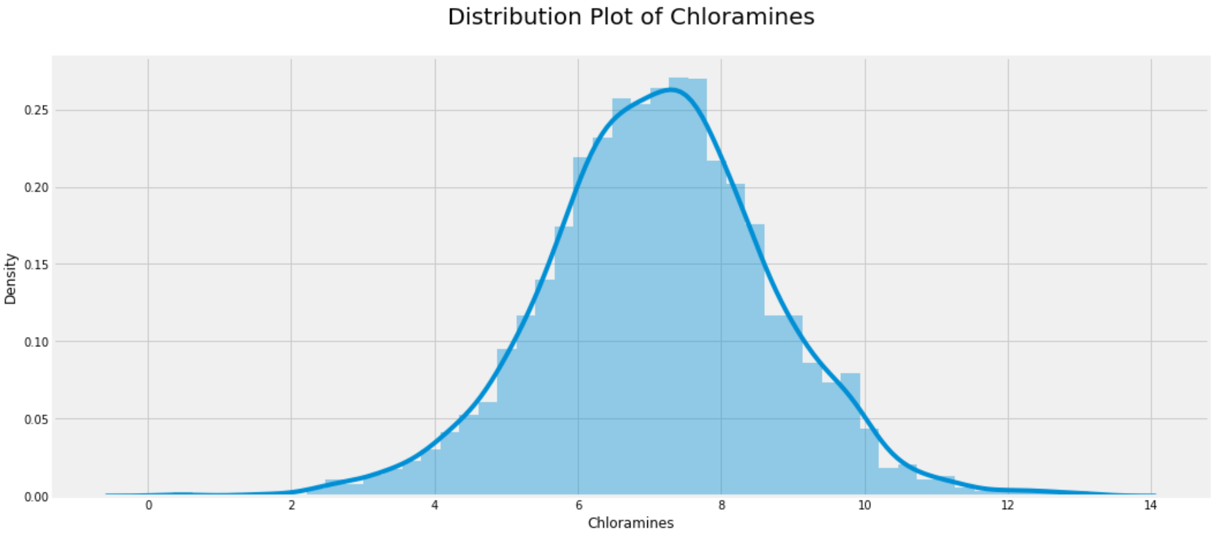
Chloramines

In [33]:
```
data['Chloramines'].describe()
```

Out[33]:
```
count    3276.000000
mean        7.122277
std         1.583085
min         0.352000
25%         6.127421
50%         7.130299
75%         8.114887
max        13.127000
Name: Chloramines, dtype: float64
```

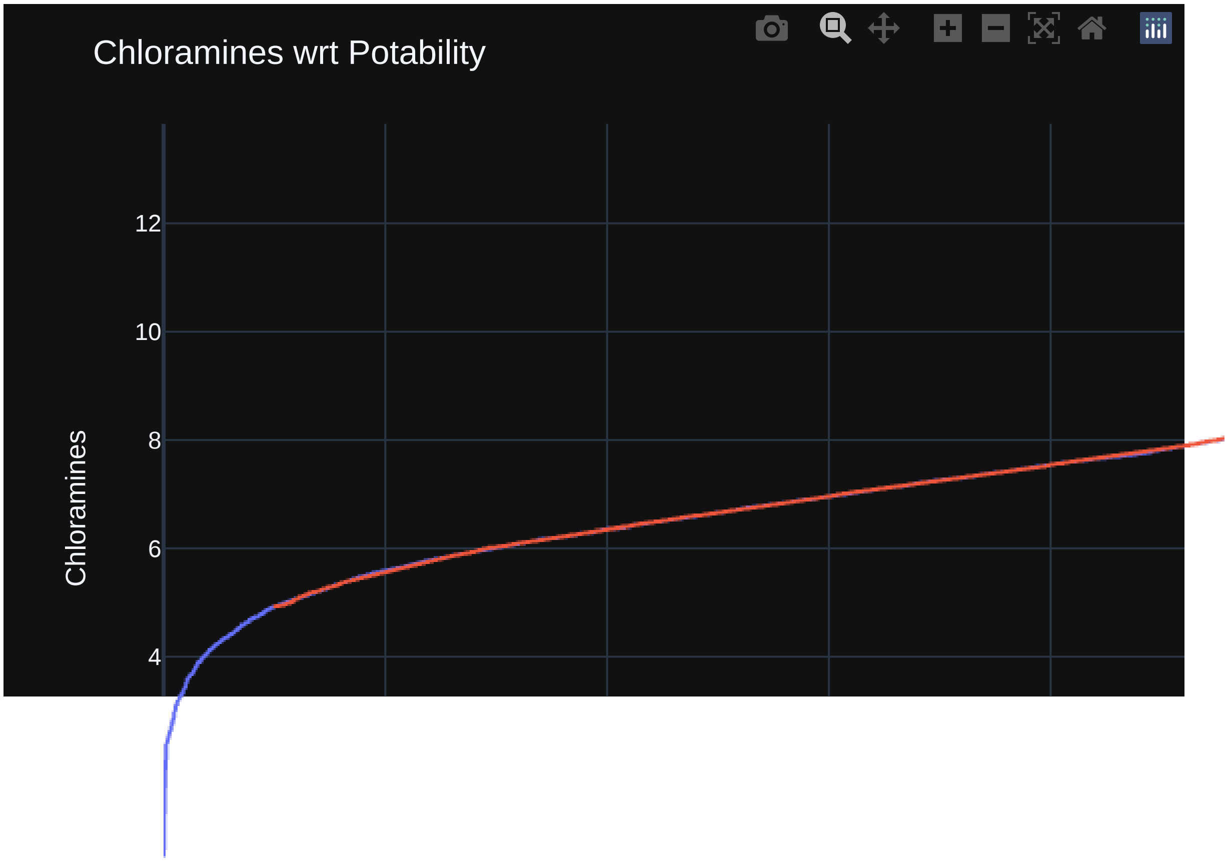
In [34]:
```
plt.figure(figsize = (16, 7))
sns.distplot(data['Chloramines'])
plt.title('Distribution Plot of Chloramines\n', fontsize =  20)
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



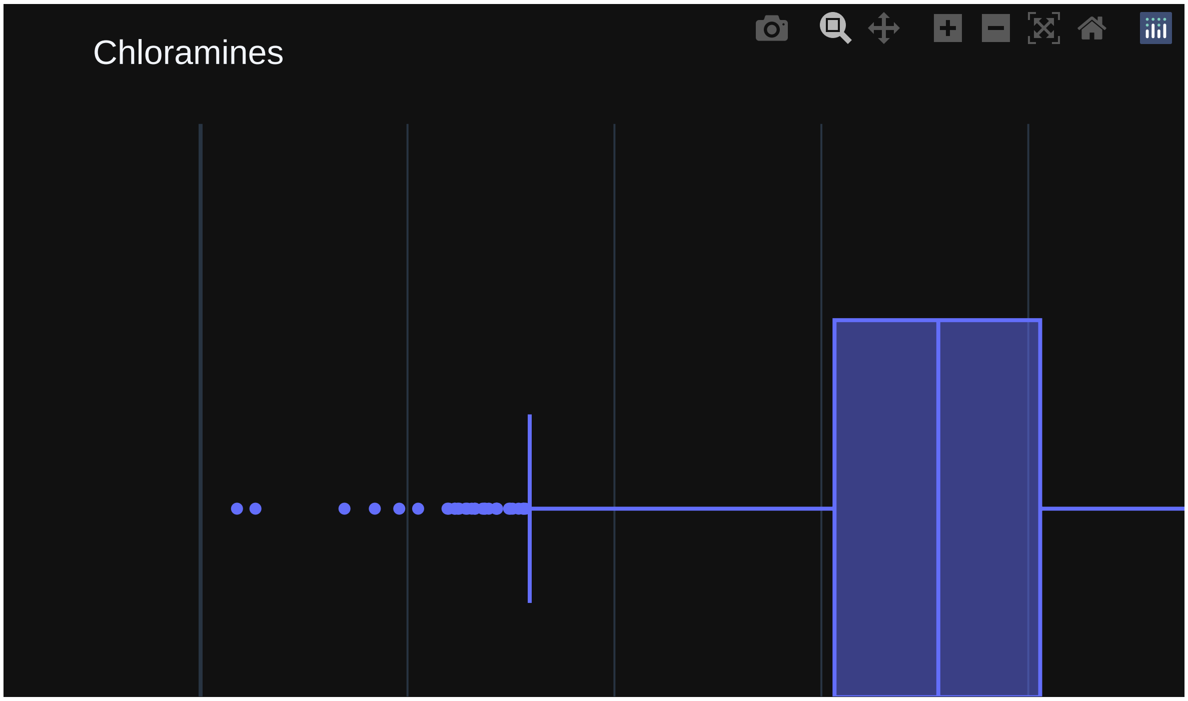
```
In [35]: fig = px.line(x=range(data['Chloramines'].count()), y=sorted(data['Chloramir
                              'x': "Count",
                              'y': "Chloramines",
                              'color':'Potability'

                          }, template = 'plotly_dark')
         fig.update_layout(title='Chloramines wrt Potability')
         fig.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [36]:

```
fig = px.box(x = 'Chloramines', data_frame = data, template = 'plotly_dark')
fig.update_layout(title='Chloramines')
fig.show()
```



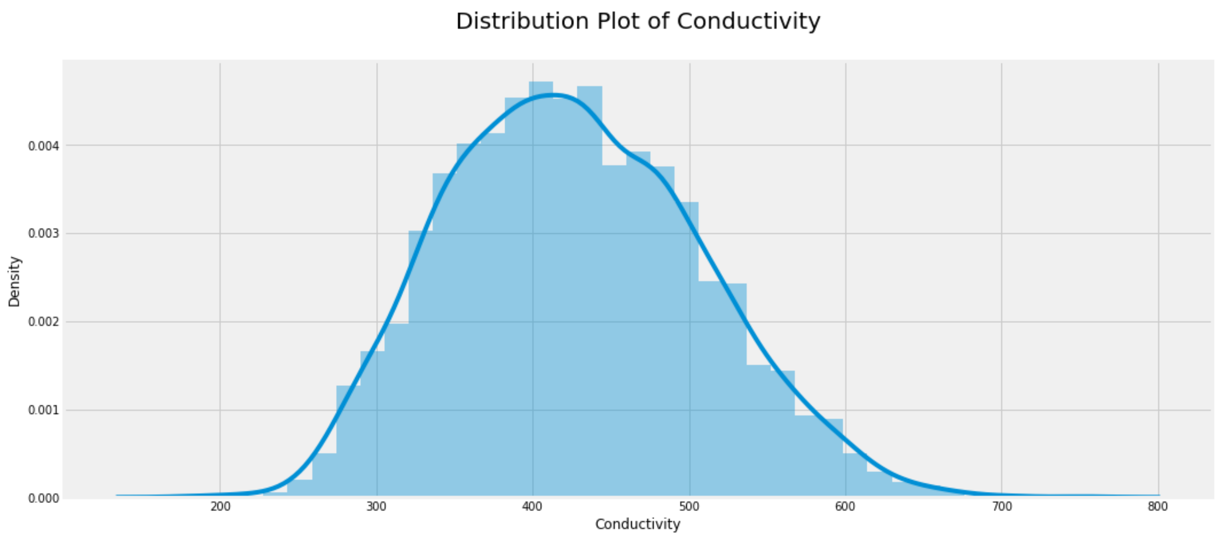
Conductivity

```
In [37]: data["Conductivity"].describe()
```

```
Out[37]: count    3276.000000
         mean      426.205111
         std        80.824064
         min       181.483754
         25%       365.734414
         50%       421.884968
         75%       481.792304
         max       753.342620
         Name: Conductivity, dtype: float64
```

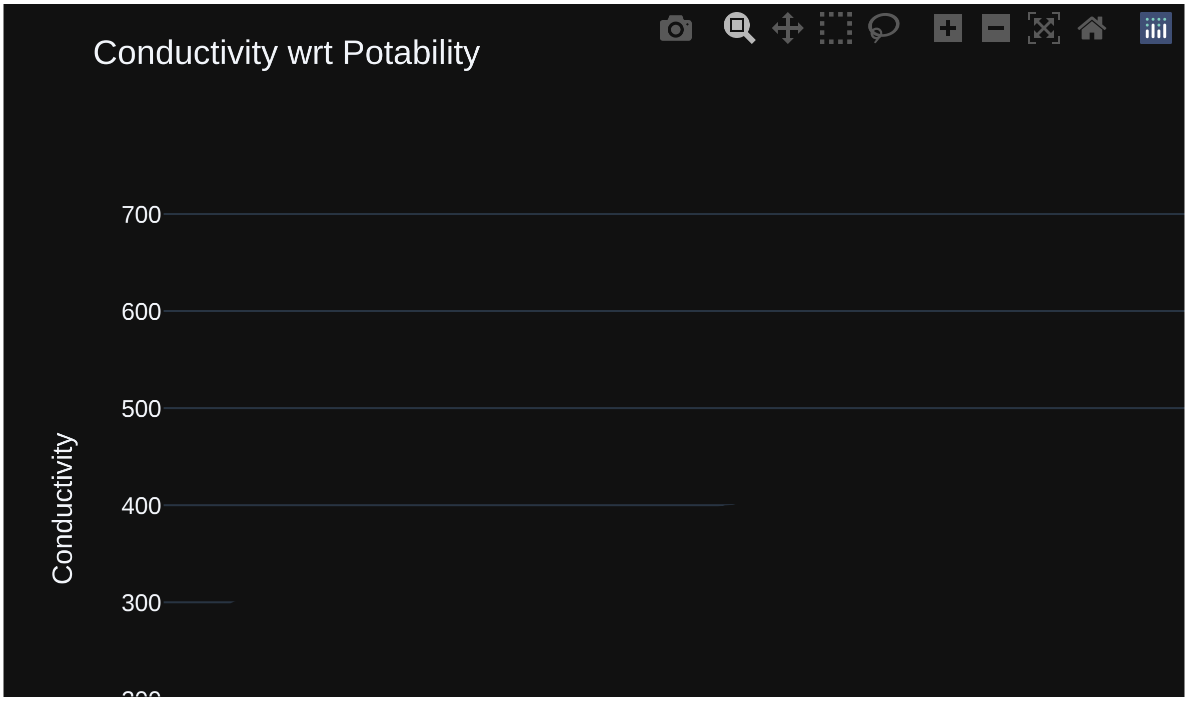
```
In [38]: plt.figure(figsize = (16, 7))
         sns.distplot(data['Conductivity'])
         plt.title('Distribution Plot of Conductivity\n', fontsize =  20)
         plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



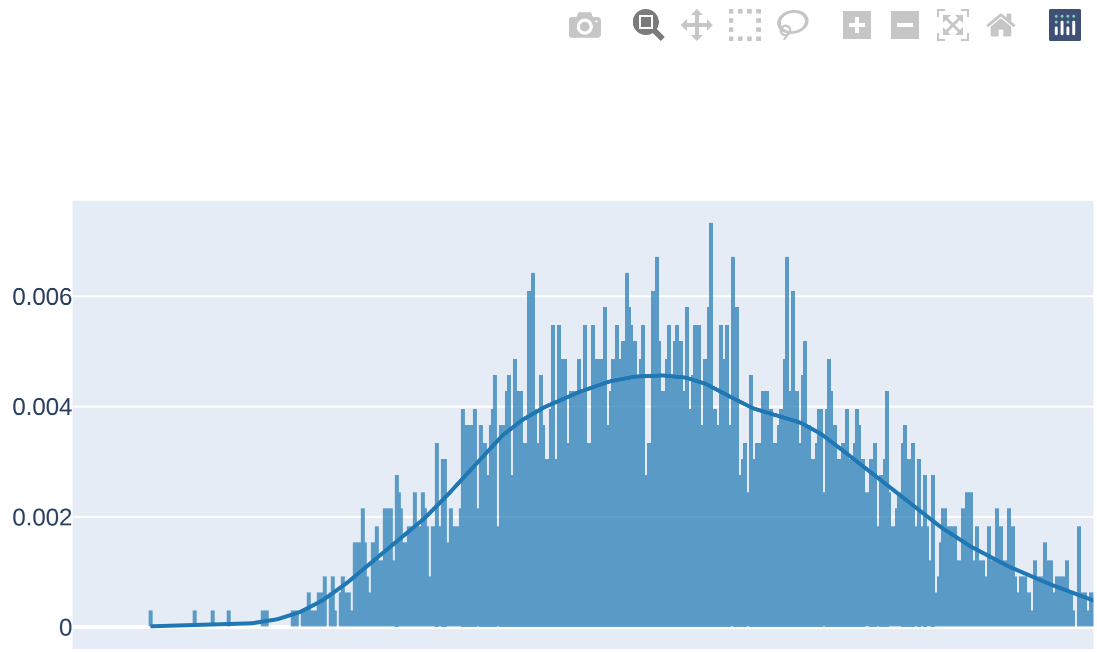
In [39]:
```
fig = px.bar(data, x=range(data['Conductivity'].count()),
             y=sorted(data['Conductivity']), labels={
                     'x': "Count",
                     'y': "Conductivity",
                     'color':'Potability'

                 },
             color=data['Potability']
             ,template = 'plotly_dark')
fig.update_layout(title='Conductivity wrt Potability')
fig.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [40]:
```python
group_labels = ['distplot'] # name of the dataset

fig = ff.create_distplot([data['Conductivity']], group_labels)
fig.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

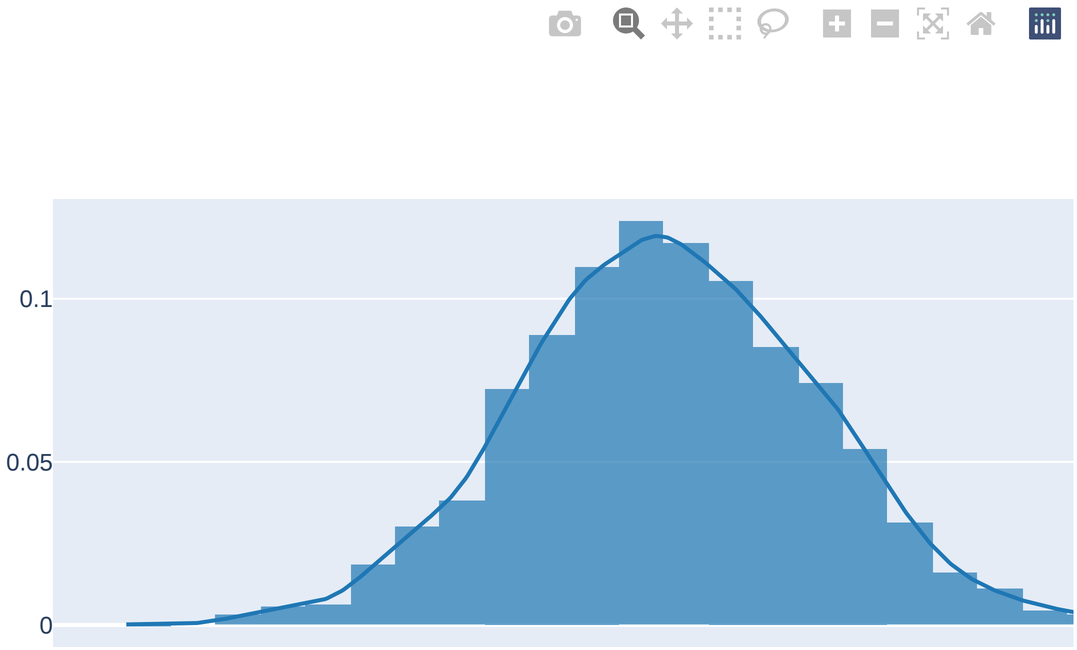


Organic_carbon

```
In [41]: data['Organic_carbon'].describe()
```

```
Out[41]: count    3276.000000
         mean       14.284970
         std         3.308162
         min         2.200000
         25%        12.065801
         50%        14.218338
         75%        16.557652
         max        28.300000
         Name: Organic_carbon, dtype: float64
```

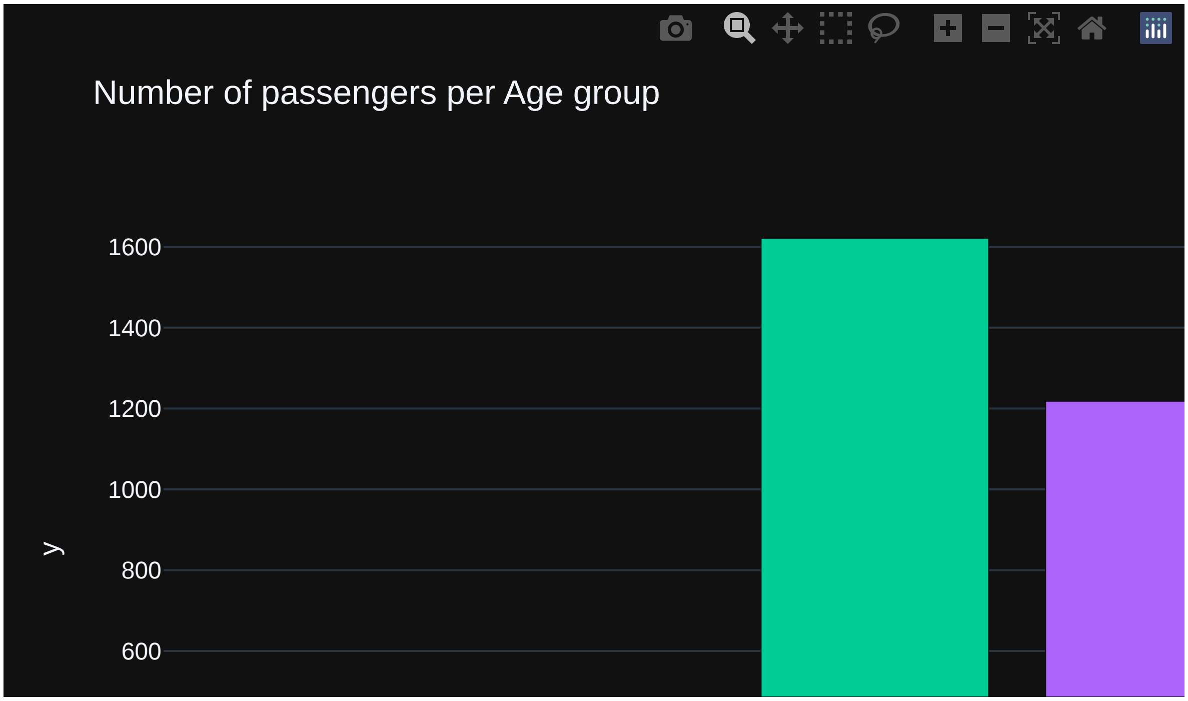
```
In [42]: group_labels = ['Organic_carbon'] # name of the dataset

         fig = ff.create_distplot([data['Organic_carbon']], group_labels)
         fig.show()
```



In [43]:

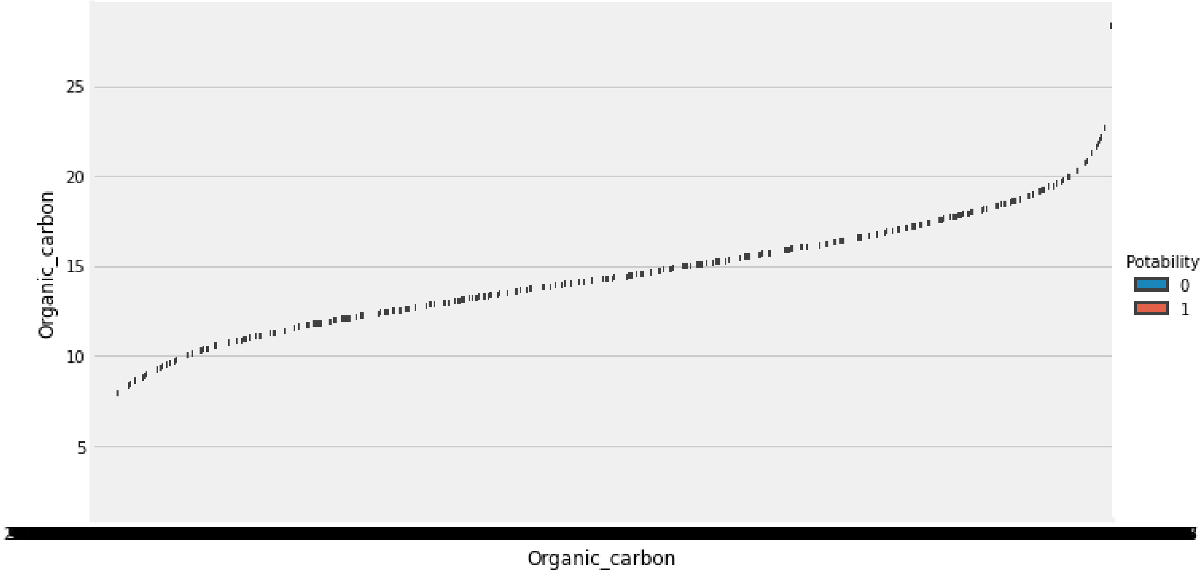
```
dt_5=data[data['Organic_carbon']<5]
dt_5_10=data[(data['Organic_carbon']>5)&(data['Organic_carbon']<10)]
dt_10_15=data[(data['Organic_carbon']>10)&(data['Organic_carbon']<15)]
dt_15_20=data[(data['Organic_carbon']>15)&(data['Organic_carbon']<20)]
dt_20_25=data[(data['Organic_carbon']>20)&(data['Organic_carbon']<25)]
dt_25=data[(data['Organic_carbon']>25)]

x_Age = ['5', '5-10', '10-15', '15-20', '25+']
y_Age = [len(dt_5.values), len(dt_5_10.values), len(dt_10_15.values), len(dt
     len(dt_25.values)]

px.bar(data_frame = data, x = x_Age, y = y_Age, color = x_Age, template = 'p
       title = 'Number of passengers per Age group')
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [44]:

```
sns.catplot(x = 'Organic_carbon', y = 'Organic_carbon', hue = 'Potability',
            height = 5, aspect = 2)
plt.show()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Turbidity

In [45]:

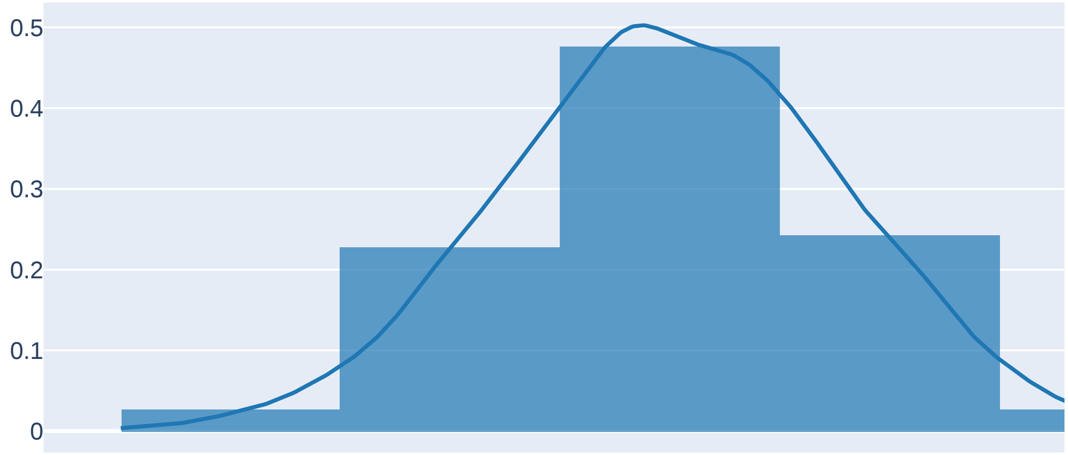
```
data['Turbidity'].describe()
```

Out[45]:

```
count    3276.000000
mean        3.966786
std         0.780382
min         1.450000
25%         3.439711
50%         3.955028
75%         4.500320
max         6.739000
Name: Turbidity, dtype: float64
```

In [46]:

```
group_labels = ['Turbidity'] # name of the dataset

fig = ff.create_distplot([data['Turbidity']], group_labels)
fig.show()
```



In [47]:

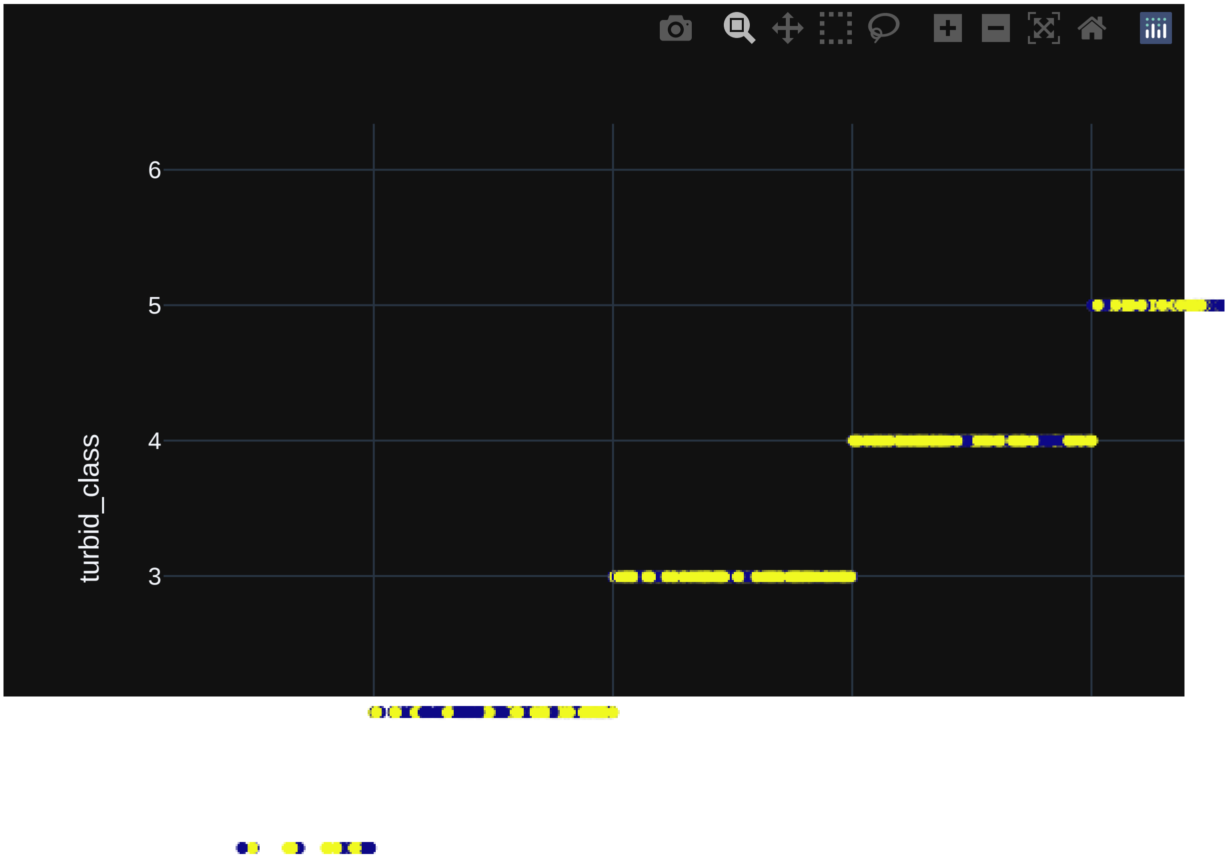
```
data['turbid_class']=data['Turbidity'].astype(int)
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
In [48]: data['turbid_class'].unique()
```

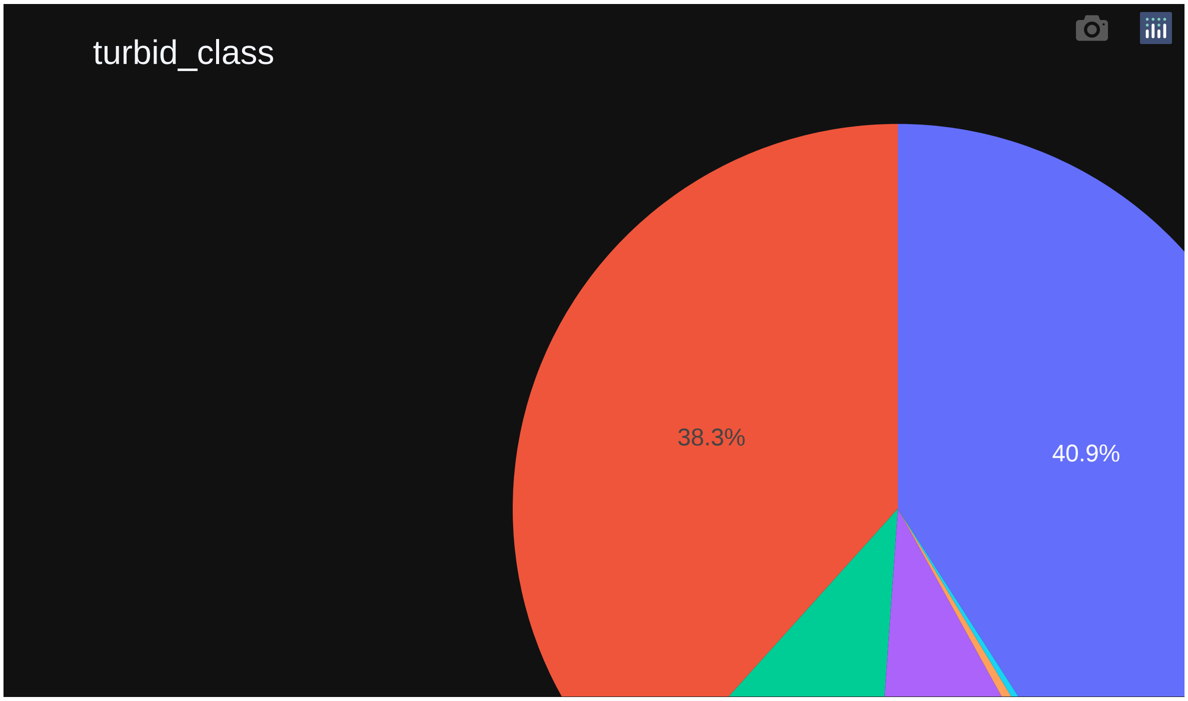
```
Out[48]: array([2, 4, 3, 5, 6, 1])
```

```
In [49]: px.scatter(data_frame = data, x = 'Turbidity', y = 'turbid_class', color = '
```



```
In [50]: fig = px.pie(data,
                      values=data['turbid_class'].value_counts(),
                      names=data['turbid_class'].value_counts().keys(),
                     )
         fig.update_layout(
             title='turbid_class',
             template = 'plotly_dark'
         )
         fig.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



```
In [51]: data=data.drop(['turbid_class'],axis=1)
```

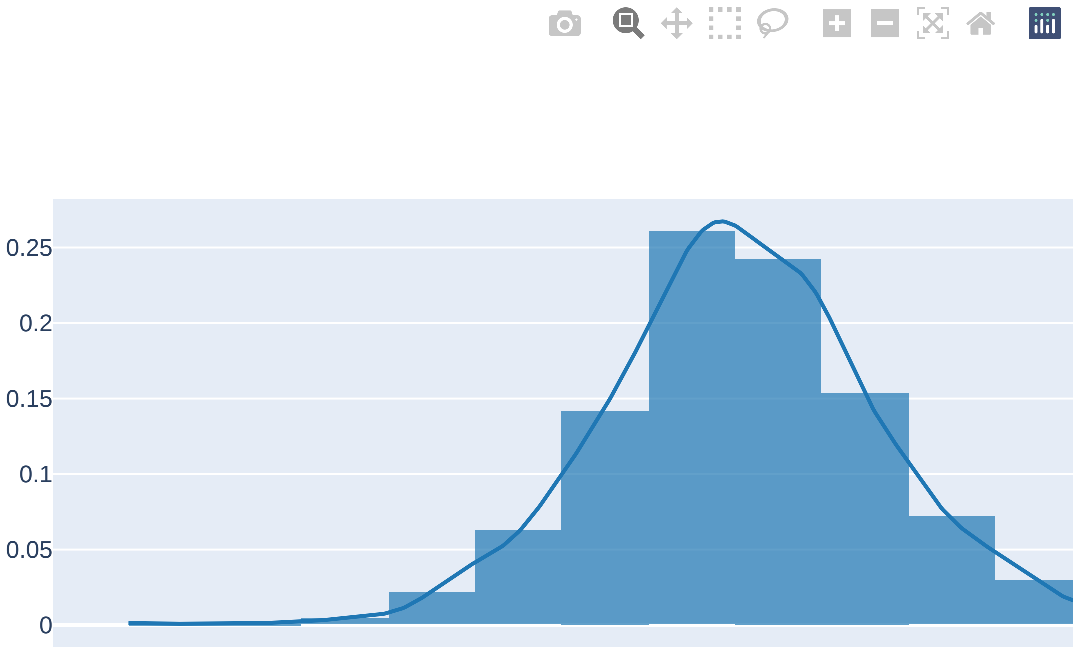
ph_random

```
In [52]: data['ph_random'].describe()
```

```
Out[52]: count    3276.000000
         mean        7.071639
         std         1.607991
         min         0.000000
         25%         6.081460
         50%         7.029490
         75%         8.063147
         max        14.000000
         Name: ph_random, dtype: float64
```

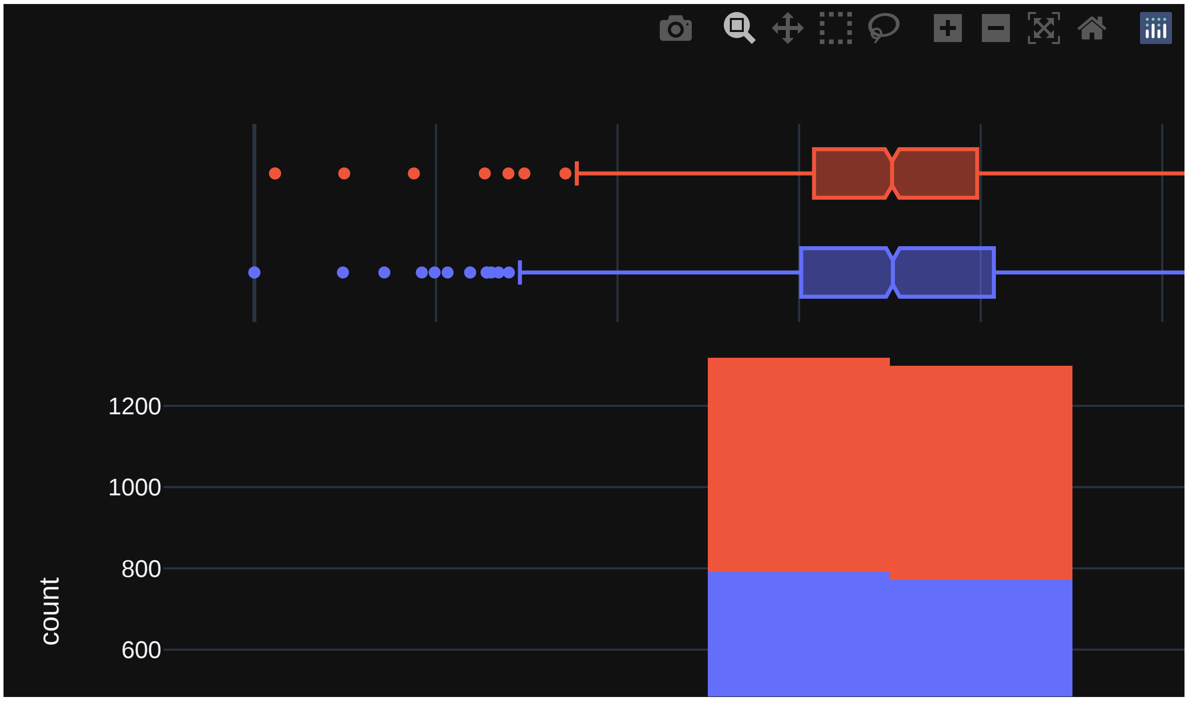
```
In [53]: group_labels = ['ph_random'] # name of the dataset

         fig = ff.create_distplot([data['ph_random']], group_labels)
         fig.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



```
In [54]: px.histogram(data_frame = data, x = 'ph_random', nbins = 10, color = 'Potabi
                      template = 'plotly_dark')
```

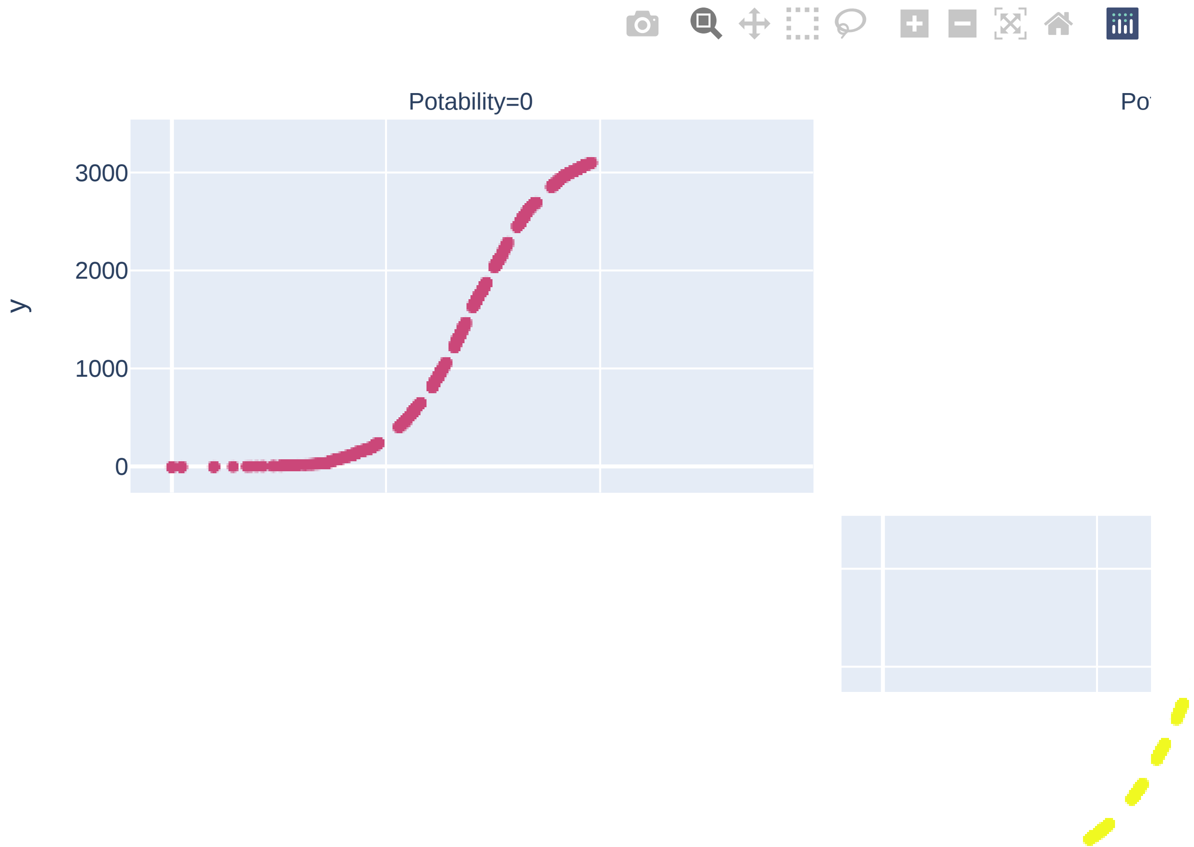
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [55]:

```
fig = px.scatter(data, sorted(data["ph_random"]), range(data["ph_random"].co
                 facet_row="Potability")
fig.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



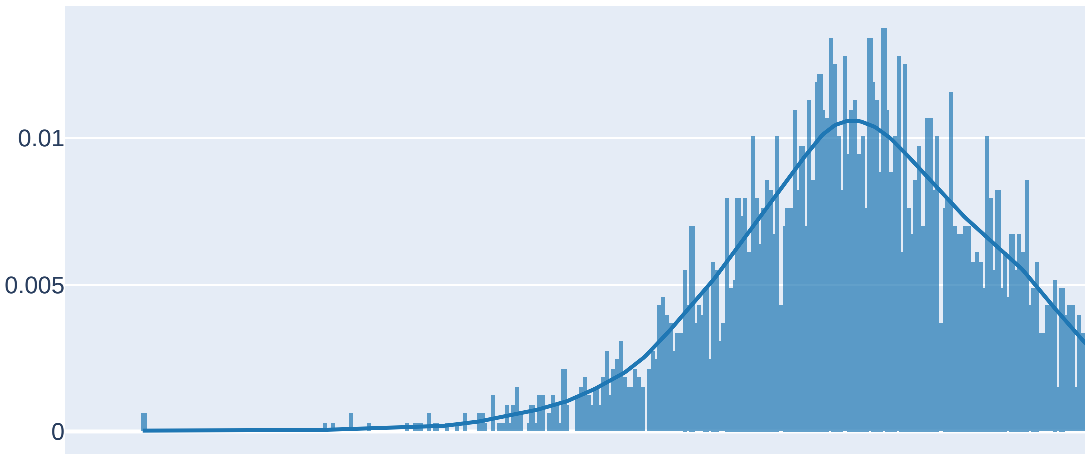
Sulfate_random

In [56]:
```
data['Sulfate_random'].describe()
```

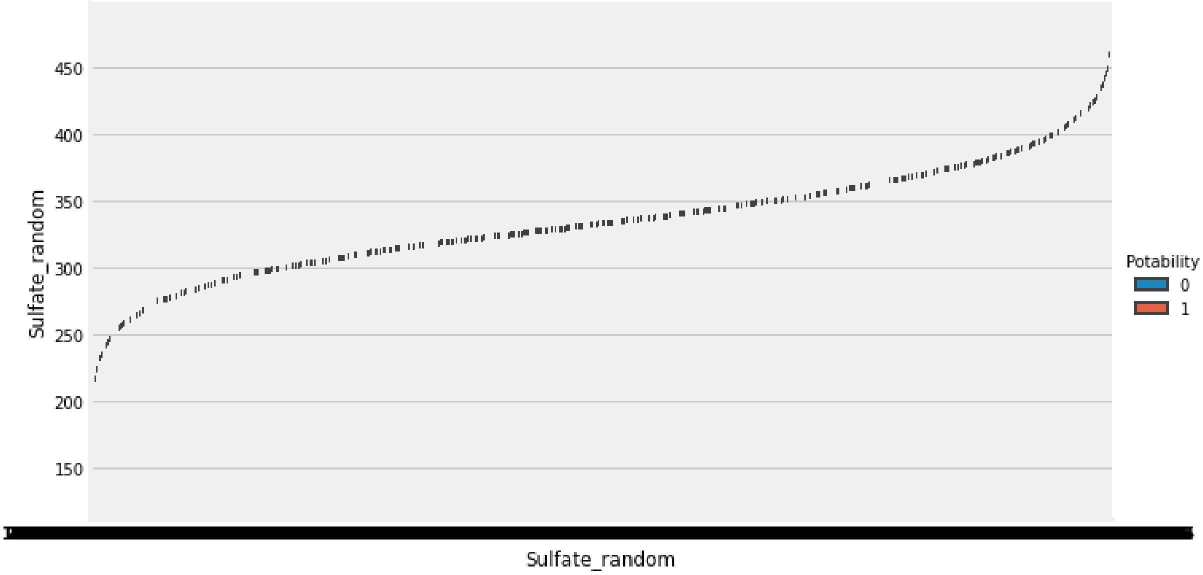
Out[56]:
```
count    3276.000000
mean      333.430954
std        41.026947
min       129.000000
25%       307.523159
50%       332.879578
75%       359.710517
max       481.030642
Name: Sulfate_random, dtype: float64
```

In [57]:
```
group_labels = ['distplot'] # name of the dataset

fig = ff.create_distplot([data['Sulfate_random']], group_labels)
fig.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



In [58]:
```
sns.catplot(x = 'Sulfate_random', y = 'Sulfate_random', hue = 'Potability',
            height = 5, aspect = 2)
plt.show()
```



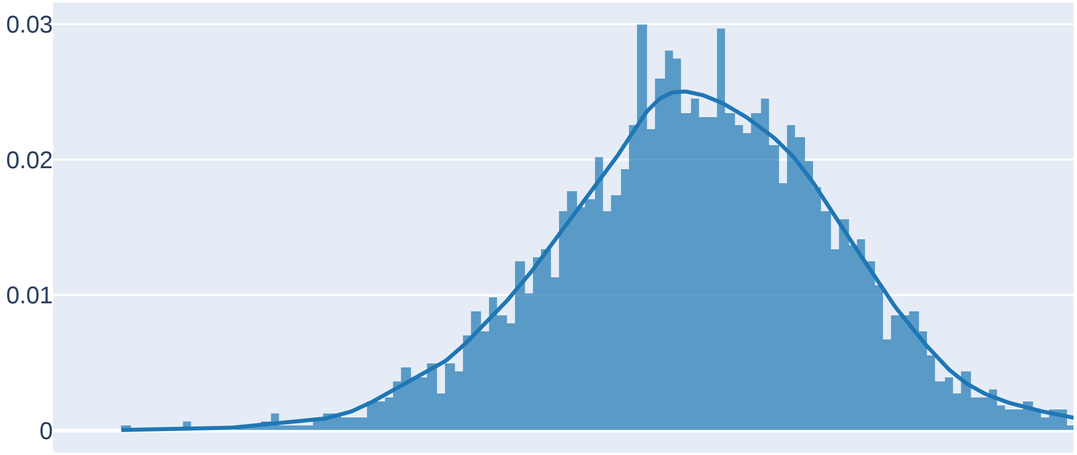
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Trihalomethanes_random

In [59]:
```
data['Trihalomethanes_random'].describe()
```

Out[59]:
```
count    3276.000000
mean       66.419200
std        16.184832
min         0.738000
25%        55.861675
50%        66.639068
75%        77.384166
max       124.000000
Name: Trihalomethanes_random, dtype: float64
```

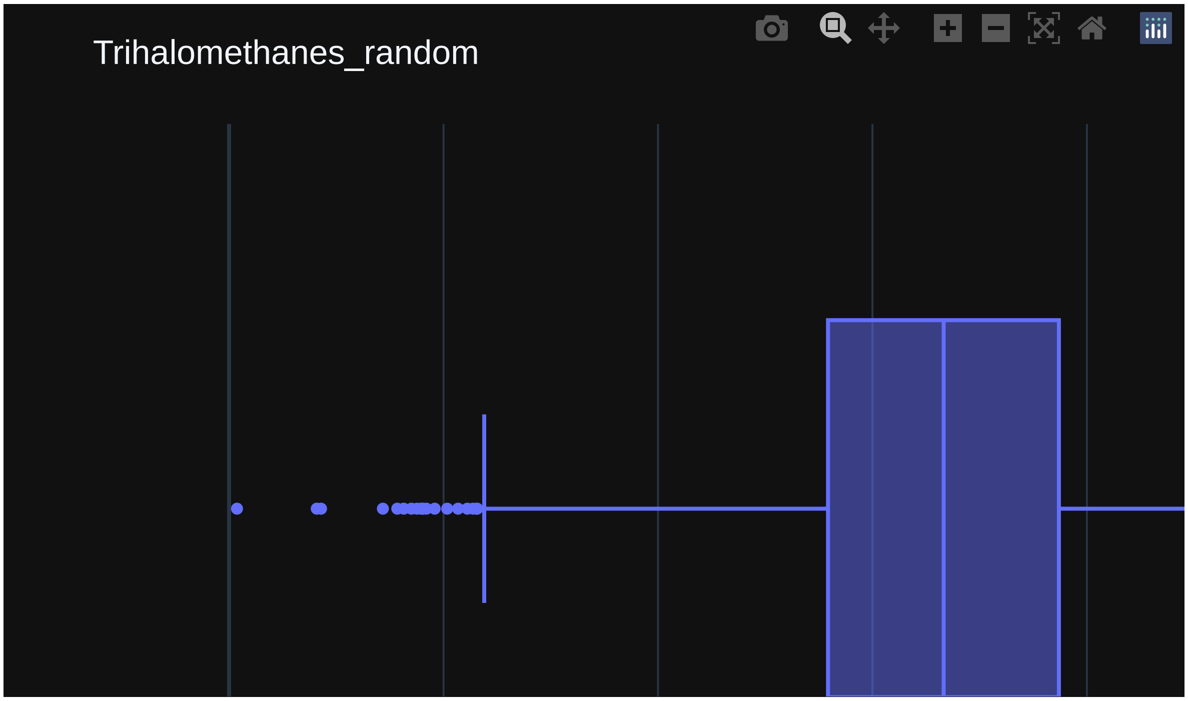
In [60]:
```
group_labels = ['Trihalomethanes_random'] # name of the dataset

fig = ff.create_distplot([data['Trihalomethanes_random']], group_labels)
fig.show()
```



In [61]:
```
fig = px.box(x = 'Trihalomethanes_random', data_frame = data, template = 'pl
fig.update_layout(title='Trihalomethanes_random')
```

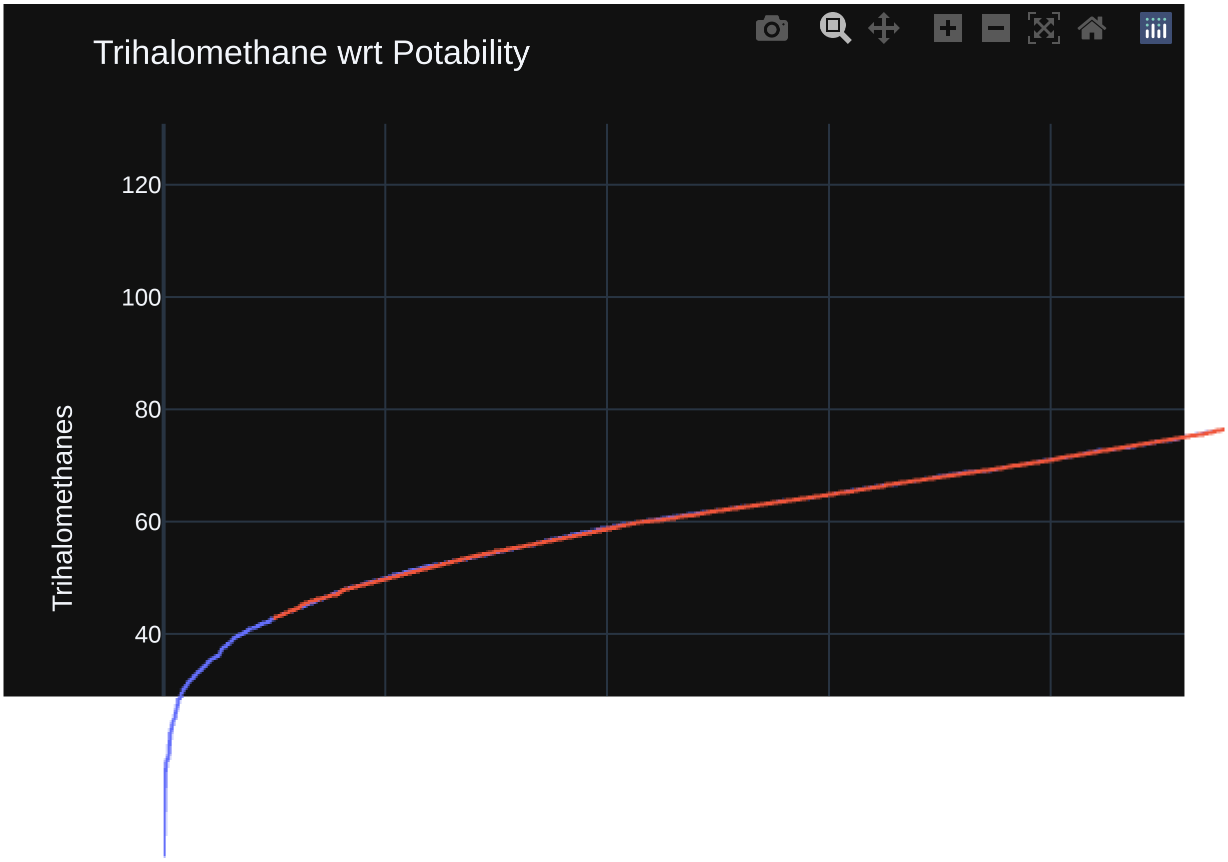
Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
fig.show()
```



```
In [62]: fig = px.line(x=range(data['Trihalomethanes_random'].count()), y=sorted(data
                             'x': "Count",
                             'y': "Trihalomethanes",
                             'color':'Potability'

                        }, template = 'plotly_dark')
         fig.update_layout(title='Trihalomethane wrt Potability')
         fig.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

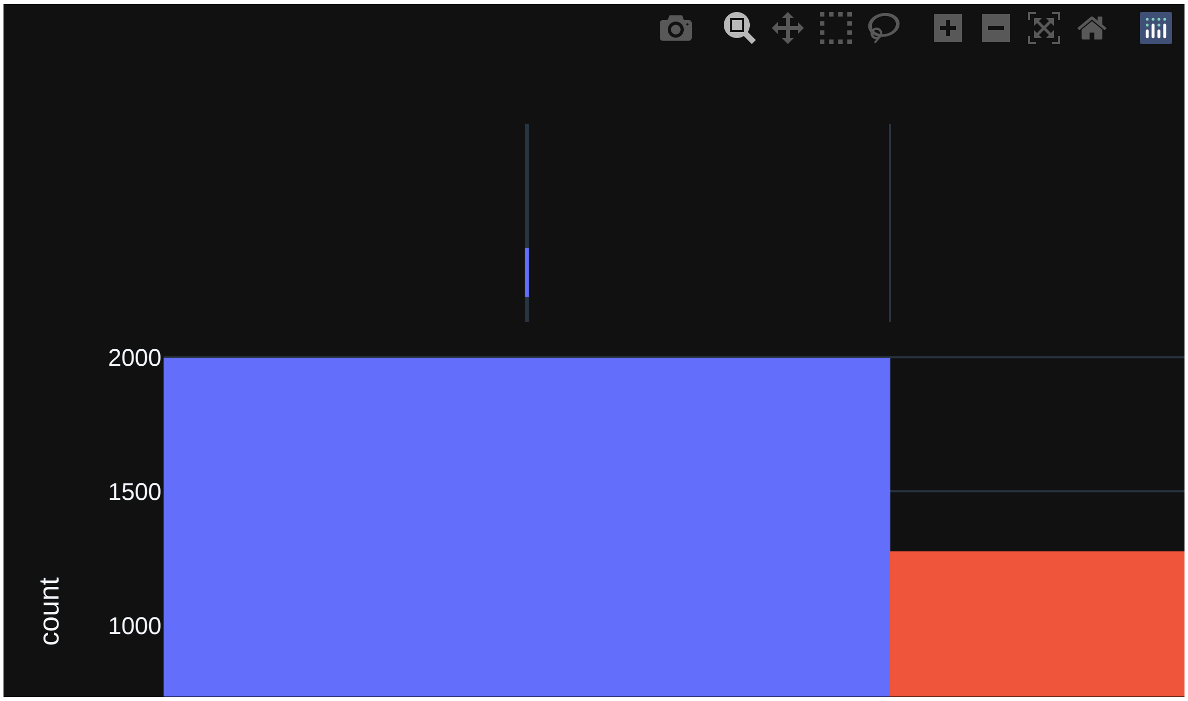


Potability

In [63]:
```
data['Potability'].describe()
```

Out[63]:
```
count    3276.000000
mean        0.390110
std         0.487849
min         0.000000
25%         0.000000
50%         0.000000
75%         1.000000
max         1.000000
Name: Potability, dtype: float64
```

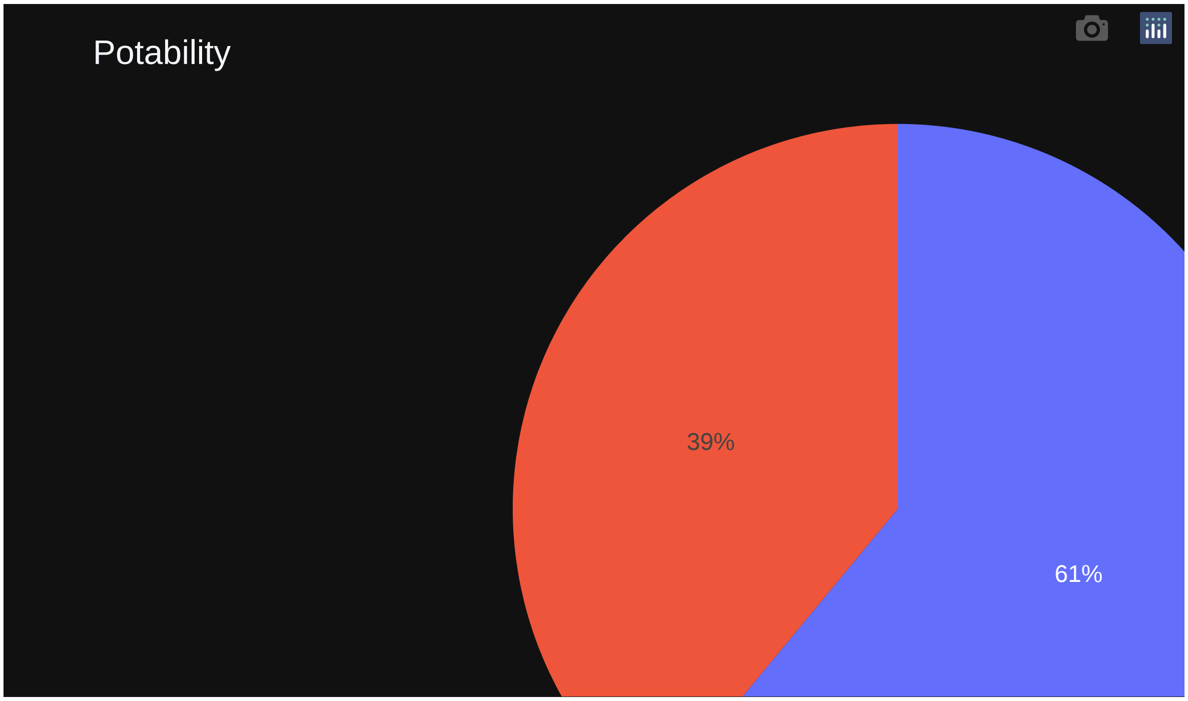
In [64]:
```
px.histogram(data_frame = data, x = 'Potability', color = 'Potability', marg
             template = 'plotly_dark')
```



In [65]:

```
fig = px.pie(data,
             values=data['Potability'].value_counts(),
             names=data['Potability'].value_counts().keys(),
            )
fig.update_layout(
    title='Potability',
    template = 'plotly_dark'
)
fig.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



Data Preprocessing

In [66]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

In [67]:

```
X=data.drop(['Potability'],axis=1)
y=data['Potability']
```

Since the data is not in a uniform shape, we scale the data using standard scalar

In [68]:

```
scaler = StandardScaler()
x=scaler.fit_transform(X)
```

In [69]:

```
# split the data to train and test set
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size=0.85,random_


print("training data shape:-{} labels{} ".format(x_train.shape,y_train.shape
print("testing data shape:-{} labels{} ".format(x_test.shape,y_test.shape))
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
training data shape:-(2784, 9) labels(2784,)
testing data shape:-(492, 9) labels(492,)
```

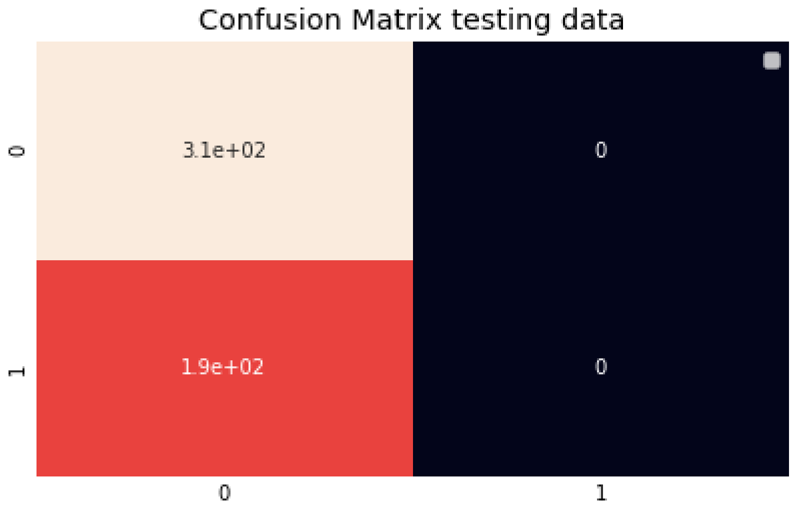
Modeling

Logistic Regression

In [70]:
```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression(random_state=0).fit(x_train, y_train)
log.score(x_test, y_test)
```

Out[70]:
```
0.6219512195121951
```

In [71]:
```
#  Confusion matrix
from sklearn.metrics import confusion_matrix
# Make Predictions
pred1=log.predict(np.array(x_test))
plt.title("Confusion Matrix testing data")
sns.heatmap(confusion_matrix(y_test,pred1),annot=True,cbar=False)
plt.legend()
plt.show()
```



K Nearest Neighbours

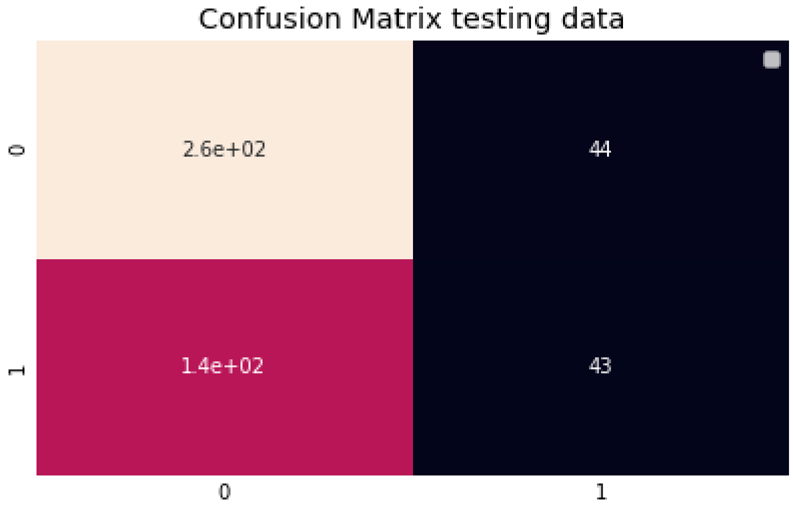
In [72]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [73]:
```
knn = KNeighborsClassifier(n_neighbors=2)
# Train the model using the training sets
knn.fit(x_train,y_train)

#Predict Output
predicted= knn.predict(x_test) # 0:Overcast, 2:Mild
```

In [74]:
```
#  Confusion matrix
fusion_matrix
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
# Make Predictions
pred1=knn.predict(np.array(x_test))
plt.title("Confusion Matrix testing data")
sns.heatmap(confusion_matrix(y_test,pred1),annot=True,cbar=False)
plt.legend()
plt.show()
```



SVM

In [75]:

```
from sklearn import svm
from sklearn.metrics import accuracy_score
```

In [76]:

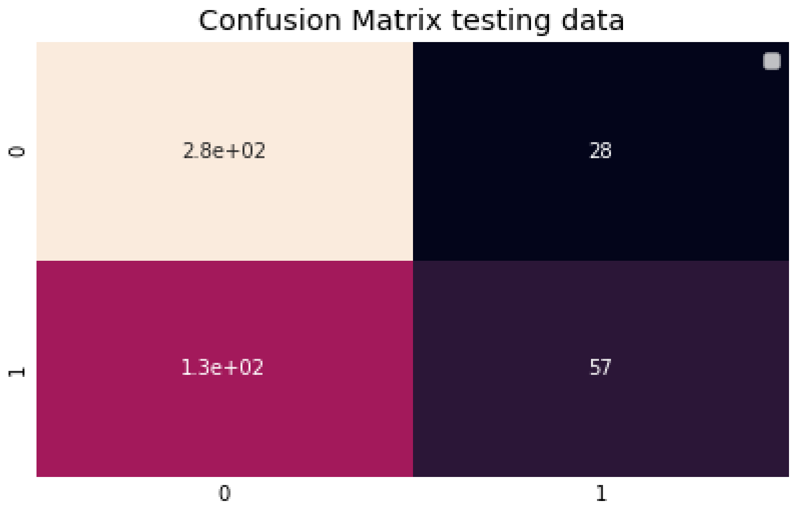
```
svmc = svm.SVC()
svmc.fit(x_train, y_train)

y_pred = svmc.predict(x_test)
print(accuracy_score(y_test,y_pred))
```

```
0.6808943089430894
```

In [77]:

```
#  Confusion matrix
from sklearn.metrics import confusion_matrix
# Make Predictions
pred1=svmc.predict(np.array(x_test))
plt.title("Confusion Matrix testing data")
sns.heatmap(confusion_matrix(y_test,pred1),annot=True,cbar=False)
plt.legend()
plt.show()
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js



Decision Tree

In [78]:

```
from sklearn import tree
from sklearn.metrics import accuracy_score
```

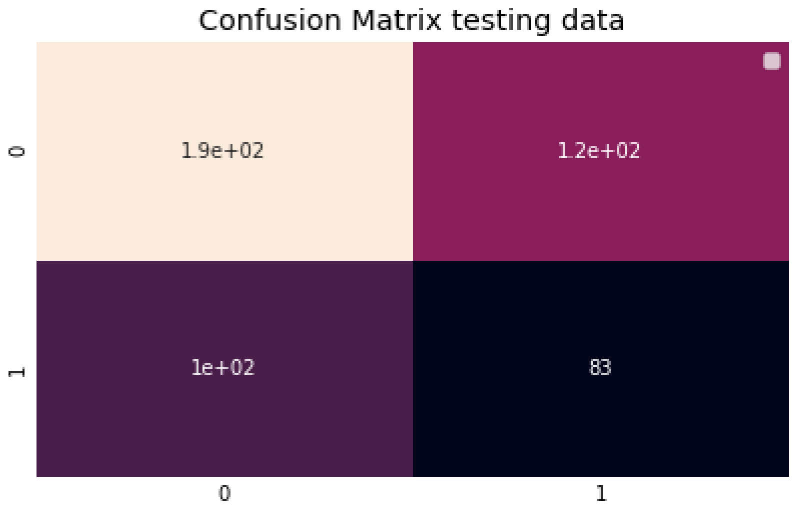
In [79]:

```
tre = tree.DecisionTreeClassifier()
tre = tre.fit(x_train, y_train)

y_pred = tre.predict(x_test)
print(accuracy_score(y_test,y_pred))
```

```
0.5487804878048781
```

In [80]:

```
#  Confusion matrix
from sklearn.metrics import confusion_matrix
# Make Predictions
pred1=tre.predict(np.array(x_test))
plt.title("Confusion Matrix testing data")
sns.heatmap(confusion_matrix(y_test,pred1),annot=True,cbar=False)
plt.legend()
plt.show()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

Random Forest

In [81]:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

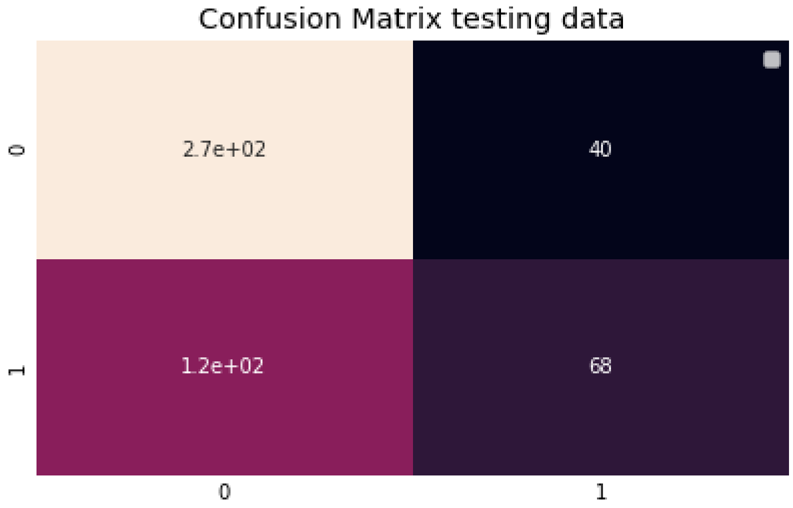
In [82]:

```
# create the model
model_rf = RandomForestClassifier(n_estimators=500, oob_score=True, random_s


# fitting the model
model_rf=model_rf.fit(x_train, y_train)

y_pred = model_rf.predict(x_test)
print(accuracy_score(y_test,y_pred))
```

```
0.6788617886178862
```

In [83]:

```
#  Confusion matrix
from sklearn.metrics import confusion_matrix
# Make Predictions
pred1=model_rf.predict(np.array(x_test))
plt.title("Confusion Matrix testing data")
sns.heatmap(confusion_matrix(y_test,pred1),annot=True,cbar=False)
plt.legend()
plt.show()
```



XG Boost

In [84]:

```
from xgboost import XGBClassifier
from sklearn.metrics import r2_score

xgb = XGBClassifier(colsample_bylevel= 0.9,
                    colsample_bytree = 0.8,
                    gamma=0.99,
                    max_depth= 5,
                    min_child_weight= 1,
                    n_estimators= 8,
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
                    nthread= 5,
                    random_state= 0,
                    )
xgb.fit(x_train,y_train)
```

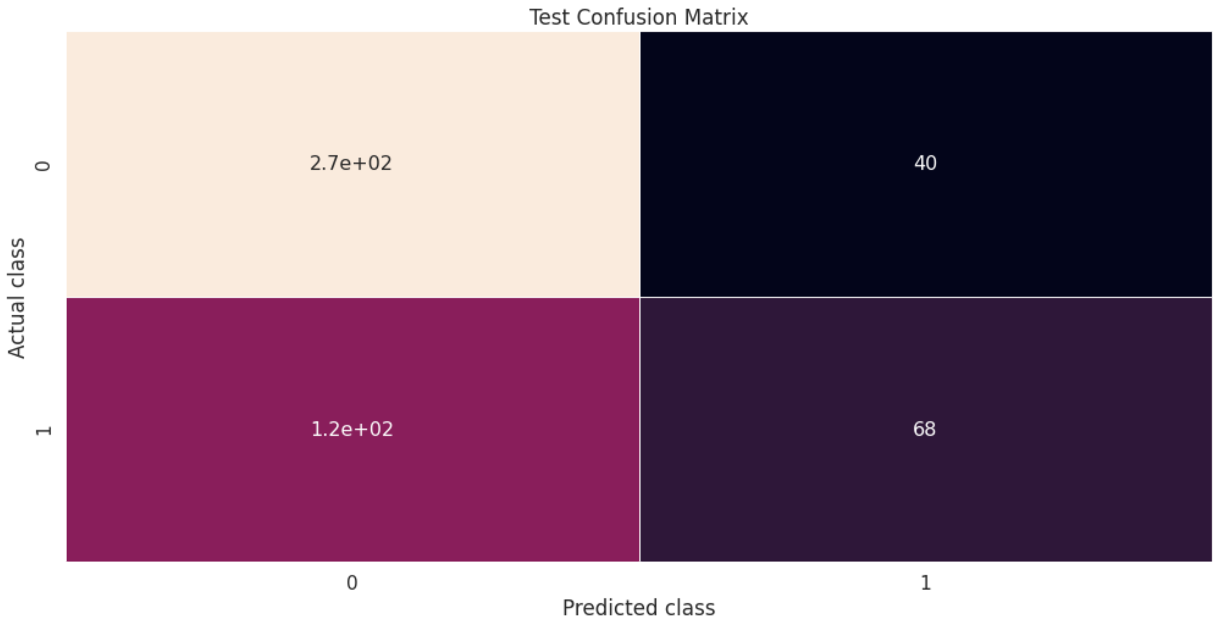
```
[14:31:40] WARNING: ../src/learner.cc:1095: Starting in XGBoost 1.3.0, the de
fault evaluation metric used with the objective 'binary:logistic' was changed
from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restor
e the old behavior.
```

```
Out[84]: XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=0.9,
                       colsample_bynode=1, colsample_bytree=0.8, gamma=0.99, gpu_id=
         -1,
                       importance_type='gain', interaction_constraints='',
                       learning_rate=0.300000012, max_delta_step=0, max_depth=5,
                       min_child_weight=1, missing=nan, monotone_constraints='()',
                       n_estimators=8, n_jobs=5, nthread=5, num_parallel_tree=1,
                       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=
         1,
                       subsample=1, tree_method='exact', validate_parameters=1,
                       verbosity=None)
```

```
In [85]: print('Accuracy of XGBoost classifier on training set: {:.2f}'
              .format(xgb.score(x_train, y_train)))
         print('Accuracy of XGBoost classifier on test set: {:.2f}'
              .format(xgb.score(x_test, y_test)))
```

```
Accuracy of XGBoost classifier on training set: 0.72
Accuracy of XGBoost classifier on test set: 0.63
```

```
In [86]: from sklearn.metrics import confusion_matrix

         conf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
         plt.figure(figsize = (15, 8))
         sns.set(font_scale=1.4) # for label size
         sns.heatmap(conf_matrix, annot=True, annot_kws={"size": 16},cbar=False, line
         plt.title("Test Confusion Matrix")
         plt.xlabel("Predicted class")
         plt.ylabel("Actual class")
         plt.savefig('conf_test.png')
         plt.show()
```



SVM tuned

In [87]:
```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
svc=SVC()
param_grid={'C':[1.2,1.5,2.2,3.5,3.2,4.1],'kernel':['linear', 'poly', 'rbf',
gridsearch=GridSearchCV(svc,param_grid=param_grid,n_jobs=-1,verbose=4,cv=3)
gridsearch.fit(x_train,y_train)
```

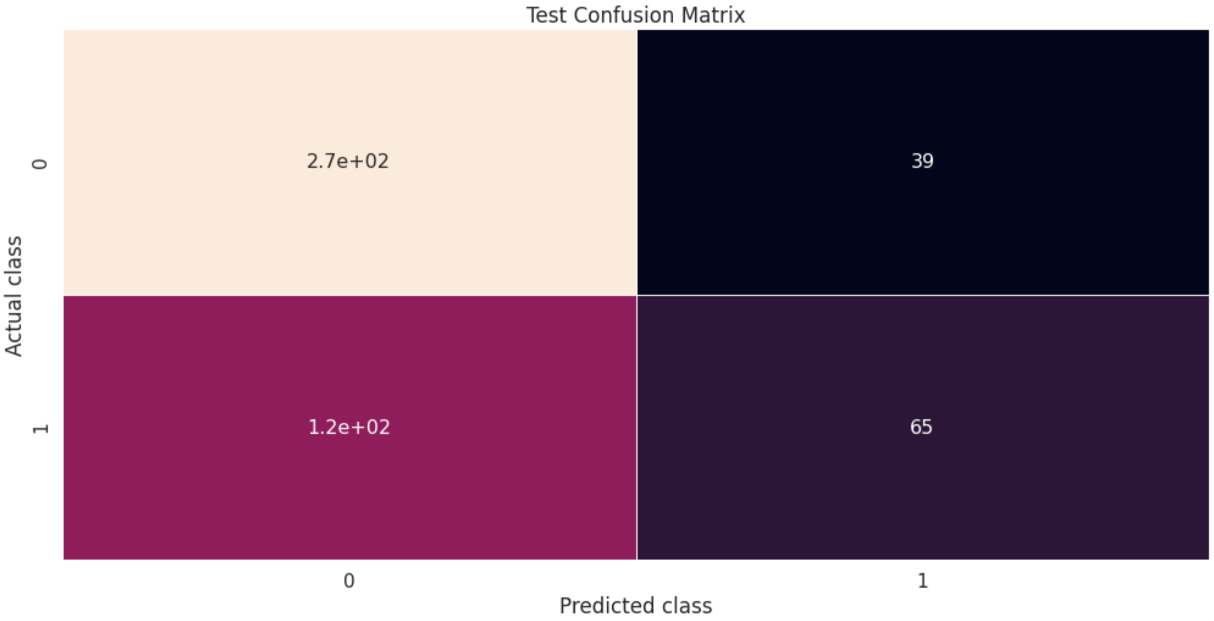
```
Fitting 3 folds for each of 240 candidates, totalling 720 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  17 tasks      | elapsed:    2.9s
[Parallel(n_jobs=-1)]: Done  90 tasks      | elapsed:    7.5s
[Parallel(n_jobs=-1)]: Done 213 tasks      | elapsed:   15.3s
[Parallel(n_jobs=-1)]: Done 384 tasks      | elapsed:   26.8s
[Parallel(n_jobs=-1)]: Done 605 tasks      | elapsed:   42.5s
[Parallel(n_jobs=-1)]: Done 720 out of 720 | elapsed:   51.0s finished
```

Out[87]:
```
GridSearchCV(cv=3, estimator=SVC(), n_jobs=-1,
             param_grid={'C': [1.2, 1.5, 2.2, 3.5, 3.2, 4.1],
                         'degree': [1, 2, 4, 8, 10], 'gamma': ['scale', 'au
to'],
                         'kernel': ['linear', 'poly', 'rbf', 'sigmoid']},
             verbose=4)
```

In [88]:
```
y_pred=gridsearch.predict(x_test)
from sklearn.metrics import confusion_matrix

conf_matrix = confusion_matrix(y_true=y_test, y_pred=y_pred)
plt.figure(figsize = (15, 8))
sns.set(font_scale=1.4) # for label size
sns.heatmap(conf_matrix, annot=True, annot_kws={"size": 16},cbar=False, line
plt.title("Test Confusion Matrix")
plt.xlabel("Predicted class")
plt.ylabel("Actual class")
```

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js

```
plt.savefig('conf_test.png')
plt.show()
```



Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js