Drinking water potability

Primeiramente importaremos e mostraremos como se comporta a base.

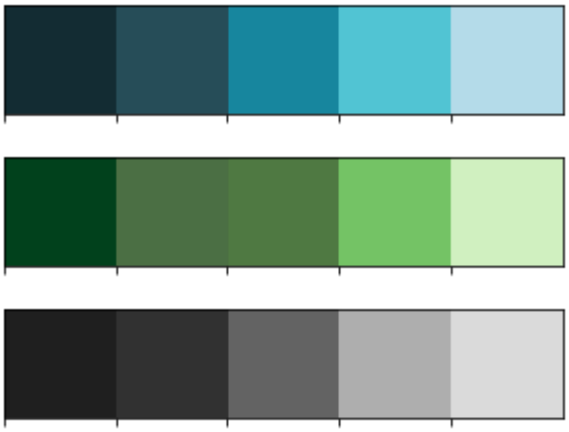
In [1]:
```
import pandas as pd

df = pd.read_csv("water_potability.csv")

df.head()
```

Out[1]:

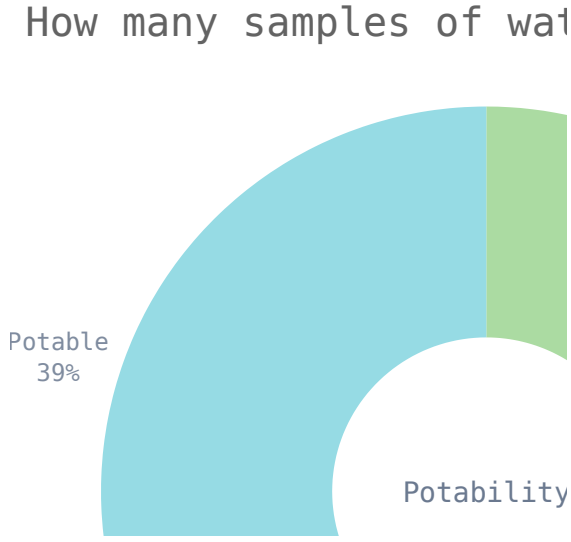
	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	T
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	

In [2]:
```
# Basic Libraries

import numpy as np
import pandas as pd
from warnings import filterwarnings
from collections import Counter

# Visualizations Libraries
import matplotlib.pyplot as plt
import seaborn as sns
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
import missingno as msno

# Data Pre-processing Libraries
from sklearn.preprocessing import StandardScaler,MinMaxScaler
from sklearn.model_selection import train_test_split

# Modelling Libraries
from sklearn.linear_model import LogisticRegression,RidgeClassifier,SGDClassi
from sklearn.linear_model import Perceptron
from sklearn.svm import SVC,LinearSVC,NuSVC
from sklearn.neighbors import KNeighborsClassifier,NearestCentroid
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,Gradie
from sklearn.naive_bayes import GaussianNB,BernoulliNB
from sklearn.ensemble import VotingClassifier

# Evaluation & CV Libraries
from sklearn.metrics import precision_score,accuracy_score
from sklearn.model_selection import RandomizedSearchCV,GridSearchCV,RepeatedS
```

In [3]:
```
colors_blue = ["#132C33", "#264D58", '#17869E', '#51C4D3', '#B4DBE9']
colors_dark = ["#1F1F1F", "#313131", '#636363', '#AEAEAE', '#DADADA']
colors_green = ['#01411C','#4B6F44','#4F7942','#74C365','#D0F0C0']
sns.palplot(colors_blue)
sns.palplot(colors_green)
sns.palplot(colors_dark)
```



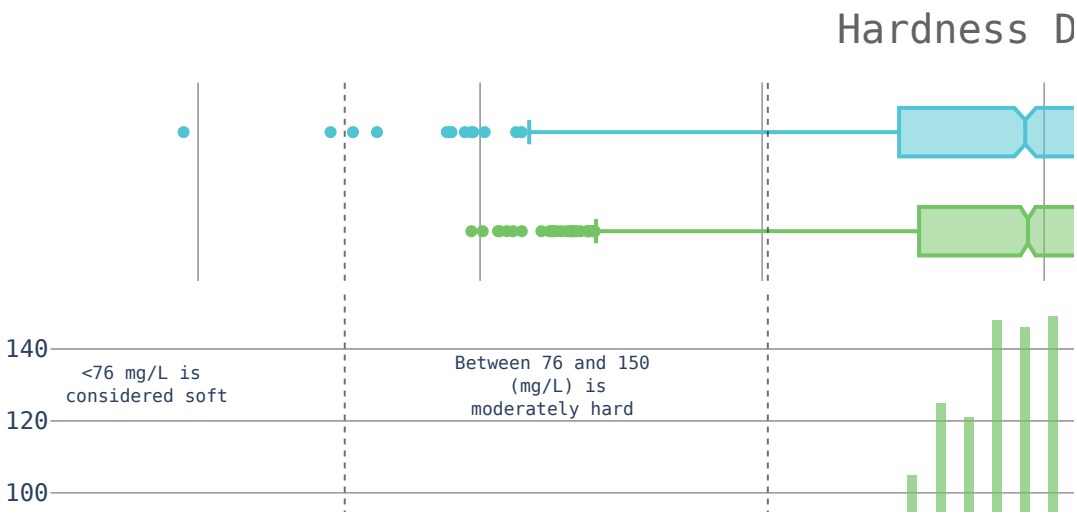
In [4]:
```
d= pd.DataFrame(df['Potability'].value_counts())
fig = px.pie(d,values='Potability',names=['Not Potable','Potable'],hole=0.4,c
             color_discrete_sequence=[colors_green[3],colors_blue[3]],
              labels={'label':'Potability','Potability':'No. Of Samples'})

fig.add_annotation(text='We can resample the data<br> to get a balanced datas
                   x=1.2,y=0.9,showarrow=False,font_size=12,opacity=0.7,font_
fig.add_annotation(text='Potability',
                   x=0.5,y=0.5,showarrow=False,font_size=14,opacity=0.7,font_

fig.update_layout(
    font_family='monospace',
    title=dict(text='Q. How many samples of water are Potable?',x=0.47,y=0.98
               font=dict(color=colors_dark[2],size=20)),
    legend=dict(x=0.37,y=-0.05,orientation='h',traceorder='reversed'),
    hoverlabel=dict(bgcolor='white'))

fig.update_traces(textposition='outside', textinfo='percent+label')

fig.show()
```



Hardenss of water: The simple definition of water hardness is the amount of dissolved calcium and magnesium in the water. Hard water is high in dissolved minerals, largely calcium and magnesium. You may have felt the effects of hard water, literally, the last time you washed your hands. Depending on the hardness of your water, after using soap to wash you may have felt like there was a film of residue left on your hands. In hard water, soap reacts with the calcium (which is relatively high in hard water) to form "soap scum". When using hard water, more soap or detergent is needed to get things clean, be it your hands, hair, or your laundry.

In [5]:

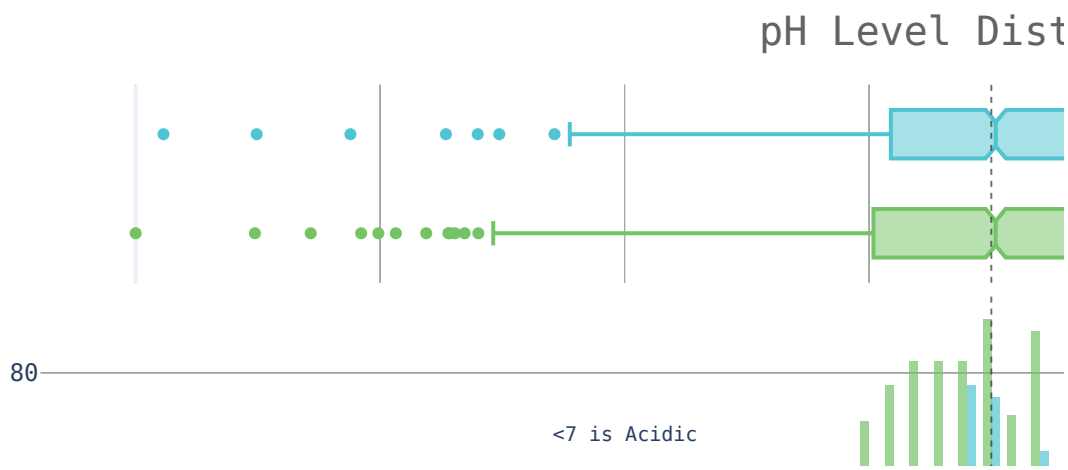
```
fig = px.histogram(df,x='Hardness',y=Counter(df['Hardness']),color='Potabilit
                   marginal='box',opacity=0.7,nbins=100,color_discrete_sequenc
                   barmode='group',histfunc='count')

fig.add_vline(x=151, line_width=1, line_color=colors_dark[1],line_dash='dot',
fig.add_vline(x=301, line_width=1, line_color=colors_dark[1],line_dash='dot',
fig.add_vline(x=76, line_width=1, line_color=colors_dark[1],line_dash='dot',c

fig.add_annotation(text='<76 mg/L is<br> considered soft',x=40,y=130,showarrc
fig.add_annotation(text='Between 76 and 150<br> (mg/L) is<br>moderately hard'
fig.add_annotation(text='Between 151 and 300 (mg/L)<br> is considered hard',x
fig.add_annotation(text='>300 mg/L is<br> considered very hard',x=340,y=130,s

fig.update_layout(
    font_family='monospace',
    title=dict(text='Hardness Distribution',x=0.53,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Hardness (mg/L)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegrou
    bargap=0.3,
)
fig.show()
```



pH level: The pH of water is a measure of the acid–base equilibrium and, in most natural waters, is controlled by the carbon dioxide–bicarbonate–carbonate equilibrium system. An increased carbon dioxide concentration will therefore lower pH, whereas a decrease will cause it to rise. Temperature will also affect the equilibria and the pH. In pure water, a decrease in pH of about 0.45 occurs as the temperature is raised by 25 °C. In water with a buffering capacity imparted by bicarbonate, carbonate and hydroxyl ions, this temperature effect is modified (APHA, 1989). The pH of most drinking-water lies within the range 6.5–8.5. Natural waters can be of lower pH, as a result of, for example, acid rain or higher pH in limestone areas.

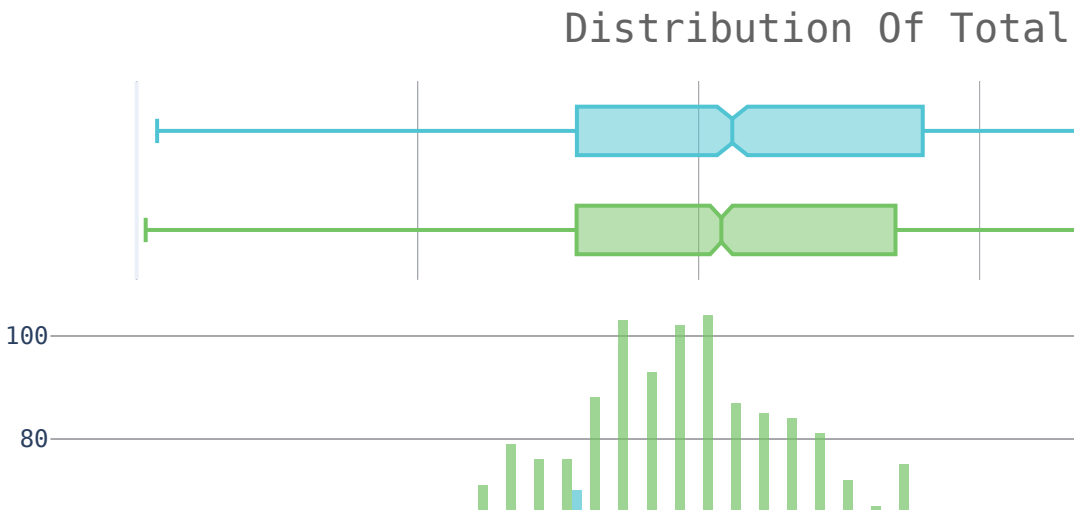
In [6]:

```
fig = px.histogram(df,x='ph',y=Counter(df['ph']),color='Potability',template=
                   marginal='box',opacity=0.7,nbins=100,color_discrete_sequenc
                   barmode='group',histfunc='count')

fig.add_vline(x=7, line_width=1, line_color=colors_dark[1],line_dash='dot',op

fig.add_annotation(text='<7 is Acidic',x=4,y=70,showarrow=False,font_size=10)
fig.add_annotation(text='>7 is Basic',x=10,y=70,showarrow=False,font_size=10)


fig.update_layout(
    font_family='monospace',
    title=dict(text='pH Level Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='pH Level',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegrou
    bargap=0.3,
)
fig.show()
```



TDS: TDS means concentration of dissolved particles or solids in water. TDS comprises of inorganic salts such as calcium, magnesium, chlorides, sulfates, bicarbonates, etc, along with many more inorganic compounds that easily dissolve in water.

In [7]:
```
fig = px.histogram(df,x='Solids',y=Counter(df['Solids']),color='Potability',t
                   marginal='box',opacity=0.7,nbins=100,color_discrete_sequenc
                   barmode='group',histfunc='count')

fig.update_layout(
    font_family='monospace',
    title=dict(text='Distribution Of Total Dissolved Solids',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Dissolved Solids (ppm)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegrou
    bargap=0.3,
)
fig.show()
```

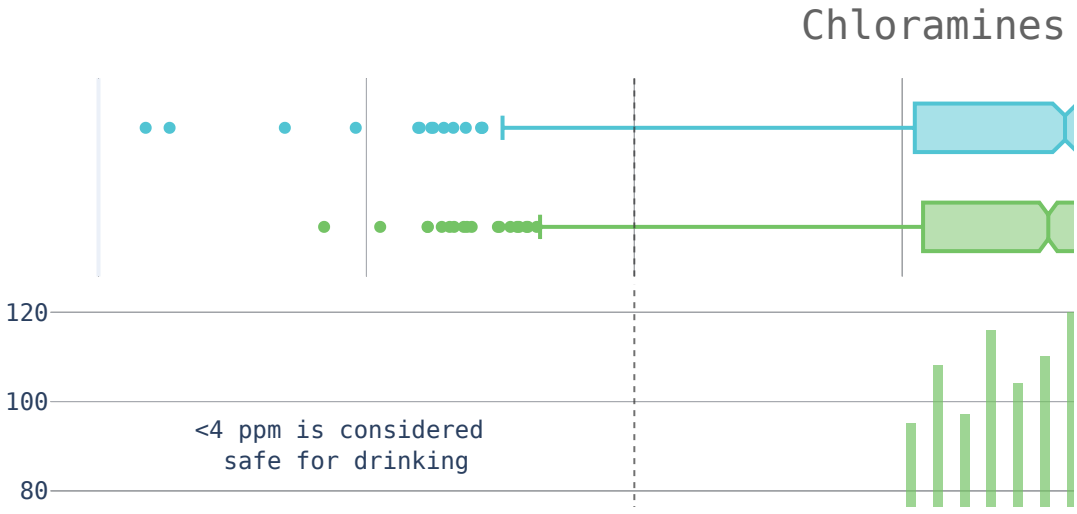


Chloramines: Chloramines (also known as secondary disinfection) are disinfectants used to treat drinking water and they:

Are most commonly formed when ammonia is added to chlorine to treat drinking water. Provide longer-lasting disinfection as the water moves through pipes to consumers. Chloramines have been used by water utilities since the 1930s.

In [8]:

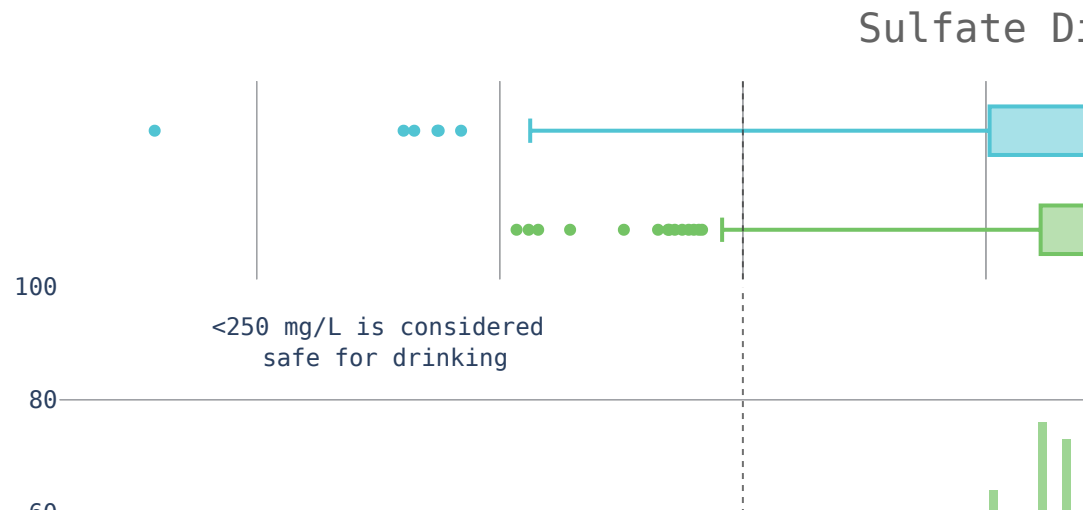
```
fig = px.histogram(df,x='Chloramines',y=Counter(df['Chloramines']),color='Pot
                   marginal='box',opacity=0.7,nbins=100,color_discrete_sequenc
                   barmode='group',histfunc='count')

fig.add_vline(x=4, line_width=1, line_color=colors_dark[1],line_dash='dot',op

fig.add_annotation(text='<4 ppm is considered<br> safe for drinking',x=1.8,y=

fig.update_layout(
    font_family='monospace',
    title=dict(text='Chloramines Distribution',x=0.53,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Chloramines (ppm)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegrou
    bargap=0.3,
)
fig.show()
```



Sulfate: Sulfate (SO4) can be found in almost all natural water. The origin of most sulfate compounds is the oxidation of sulfite ores, the presence of shales, or the industrial wastes. Sulfate is one of the major dissolved components of rain. High concentrations of sulfate in the water we drink can have a laxative effect when combined with calcium and magnesium, the two most common constituents of hardness.

In [9]:

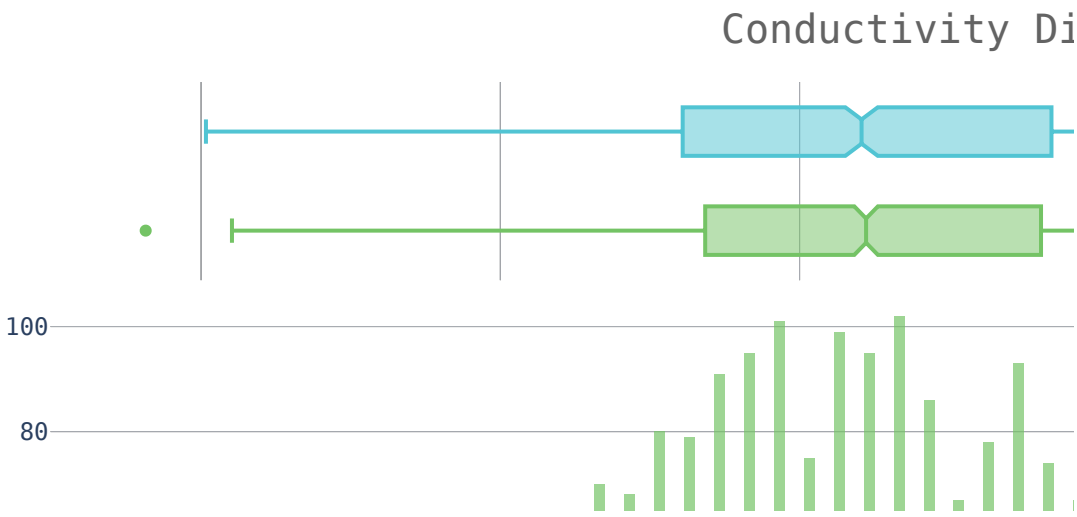
```
fig = px.histogram(df,x='Sulfate',y=Counter(df['Sulfate']),color='Potability'
                   marginal='box',opacity=0.7,nbins=100,color_discrete_sequenc
                   barmode='group',histfunc='count')

fig.add_vline(x=250, line_width=1, line_color=colors_dark[1],line_dash='dot',

fig.add_annotation(text='<250 mg/L is considered<br> safe for drinking',x=175

fig.update_layout(
    font_family='monospace',
    title=dict(text='Sulfate Distribution',x=0.53,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Sulfate (mg/L)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegrou
    bargap=0.3,
)
fig.show()
```



Conductivity: Conductivity is a measure of the ability of water to pass an electrical current. Because dissolved salts and other inorganic chemicals conduct electrical current, conductivity increases as salinity increases. Organic compounds like oil do not conduct electrical current very well and therefore have a low conductivity when in water. Conductivity is also affected by temperature: the warmer the water, the higher the conductivity.

In [10]:

```
fig = px.histogram(df,x='Conductivity',y=Counter(df['Conductivity']),color='F
                   marginal='box',opacity=0.7,nbins=100,color_discrete_sequenc
                   barmode='group',histfunc='count')

fig.add_annotation(text='The Conductivity range <br> is safe for both (200-80
                    x=600,y=90,showarrow=False)

fig.update_layout(
    font_family='monospace',
    title=dict(text='Conductivity Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Conductivity (µS/cm)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegrou
    bargap=0.3,
)
fig.show()
```

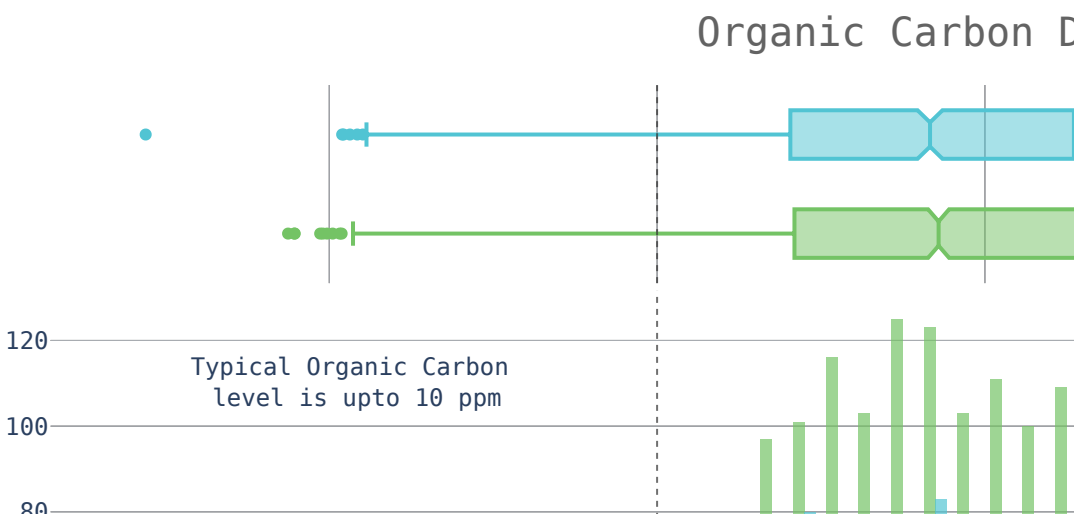


Organic Carbon: Organic contaminants (natural organic substances, insecticides, herbicides, and other agricultural chemicals) enter waterways in rainfall runoff. Domestic and industrial wastewaters also contribute organic contaminants in various amounts. As a result of accidental

spills or leaks, industrial organic wastes may enter streams. Some of the contaminants may not be completely removed by treatment processes; therefore, they could become a problem for drinking water sources. It is important to know the organic content in a waterway.

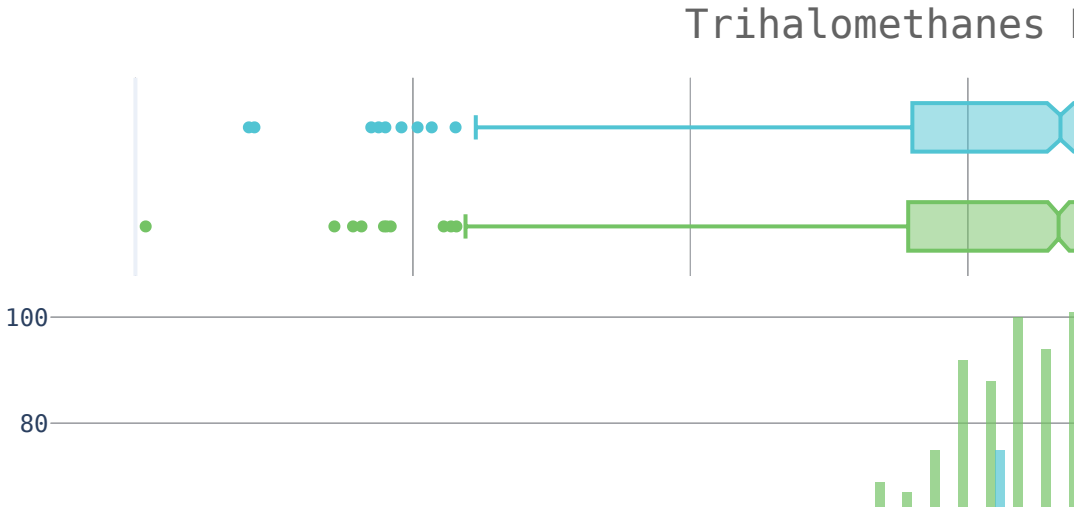
In [11]:

```
fig = px.histogram(df,x='Organic_carbon',y=Counter(df['Organic_carbon']),colo
                   marginal='box',opacity=0.7,nbins=100,color_discrete_sequenc
                   barmode='group',histfunc='count')

fig.add_vline(x=10, line_width=1, line_color=colors_dark[1],line_dash='dot',c

fig.add_annotation(text='Typical Organic Carbon<br> level is upto 10 ppm',x=5

fig.update_layout(
    font_family='monospace',
    title=dict(text='Organic Carbon Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Organic Carbon (ppm)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegrou
    bargap=0.3,
)
fig.show()
```



Trihalomethanes: Trihalomethanes (THMs) are the result of a reaction between the chlorine used for disinfecting tap water and natural organic matter in the water. At elevated levels, THMs have been associated with negative health effects such as cancer and adverse reproductive outcomes.

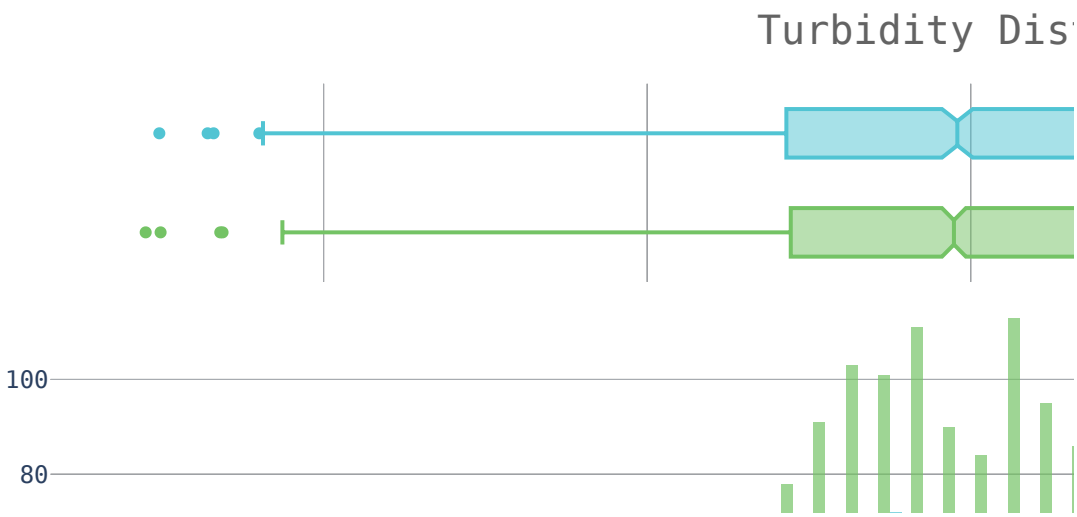
In [12]:
```
fig = px.histogram(df,x='Trihalomethanes',y=Counter(df['Trihalomethanes']),cc
                   marginal='box',opacity=0.7,nbins=100,color_discrete_sequenc
                   barmode='group',histfunc='count')

fig.add_vline(x=80, line_width=1, line_color=colors_dark[1],line_dash='dot',c

fig.add_annotation(text='Upper limit of Trihalomethanes<br> level is 80 µg/L'

fig.update_layout(
    font_family='monospace',
    title=dict(text='Trihalomethanes Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Trihalomethanes (µg/L)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegrou
    bargap=0.3,
)
fig.show()
```



Turbidity: Turbidity is the measure of relative clarity of a liquid. It is an optical characteristic of water and is a measurement of the amount of light that is scattered by material in the water when a light is shined through the water sample. The higher the intensity of scattered light, the higher the turbidity. Material that causes water to be turbid include clay, silt, very tiny inorganic and organic matter, algae, dissolved colored organic compounds, and plankton and other microscopic organisms.

In [13]:
```
fig = px.histogram(df,x='Turbidity',y=Counter(df['Turbidity']),color='Potabil
                   marginal='box',opacity=0.7,nbins=100,color_discrete_sequenc
```

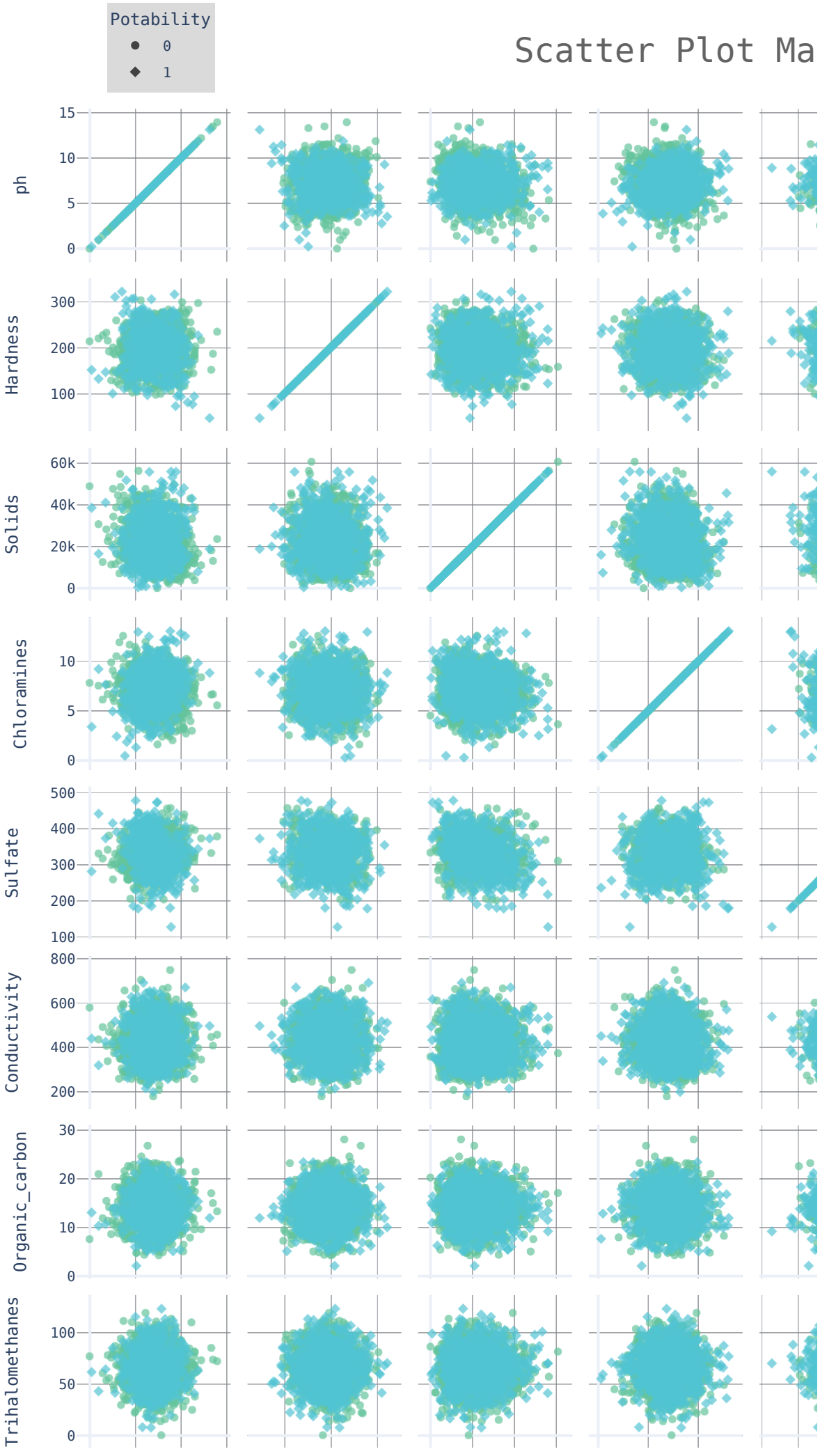
```
                  barmode='group',histfunc='count')

fig.add_vline(x=5, line_width=1, line_color=colors_dark[1],line_dash='dot',op

fig.add_annotation(text='<5 NTU Turbidity is<br> considered safe',x=6,y=90,sh

fig.update_layout(
    font_family='monospace',
    title=dict(text='Turbidity Distribution',x=0.5,y=0.95,
               font=dict(color=colors_dark[2],size=20)),
    xaxis_title_text='Turbidity (NTU)',
    yaxis_title_text='Count',
    legend=dict(x=1,y=0.96,bordercolor=colors_dark[4],borderwidth=0,tracegrou
    bargap=0.3,
)
fig.show()
```

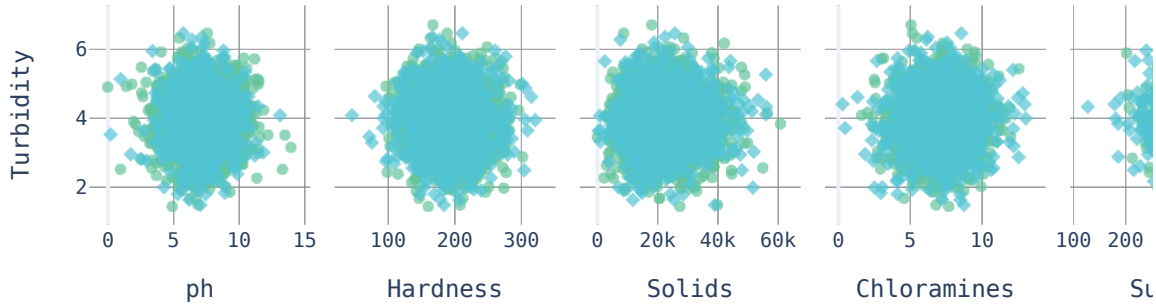


Scatter Plot Matrix helps in finding out the correlation between all the features

In [14]:

```
fig = px.scatter_matrix(df,df.drop('Potability',axis=1),height=1250,width=125
                        color_discrete_sequence=[colors_blue[3],colors_green[
                       symbol='Potability',color_continuous_scale=[colors_gre

fig.update_layout(font_family='monospace',font_size=10,
                  coloraxis_showscale=False,
                 legend=dict(x=0.02,y=1.07,bgcolor=colors_dark[4]),
                 title=dict(text='Scatter Plot Matrix b/w Features',x=0.5,y=0
                   font=dict(color=colors_dark[2],size=24)))
fig.show()
```


Potability
0
1
Scatter Plot Ma
ph
Hardness
Solids
Chloramines
Sulfate
Conductivity
Organic_carbon
Trihalomethanes
15
10
5
0
300
200
100
60k
40k
20k
0
10
5
0
500
400
300
200
100
800
600
400
200
30
20
10
0
100
50
0



As we can see, there seems to be very less correlation between all the features.

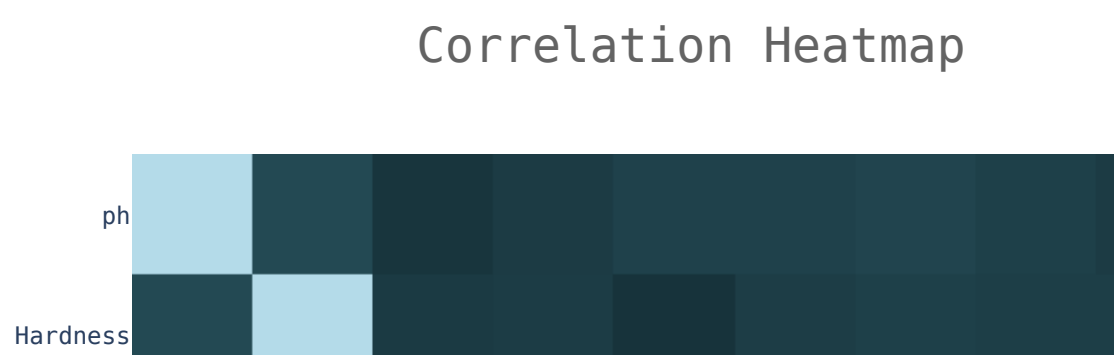
In [15]:
```
cor=df.drop('Potability',axis=1).corr()
cor
```

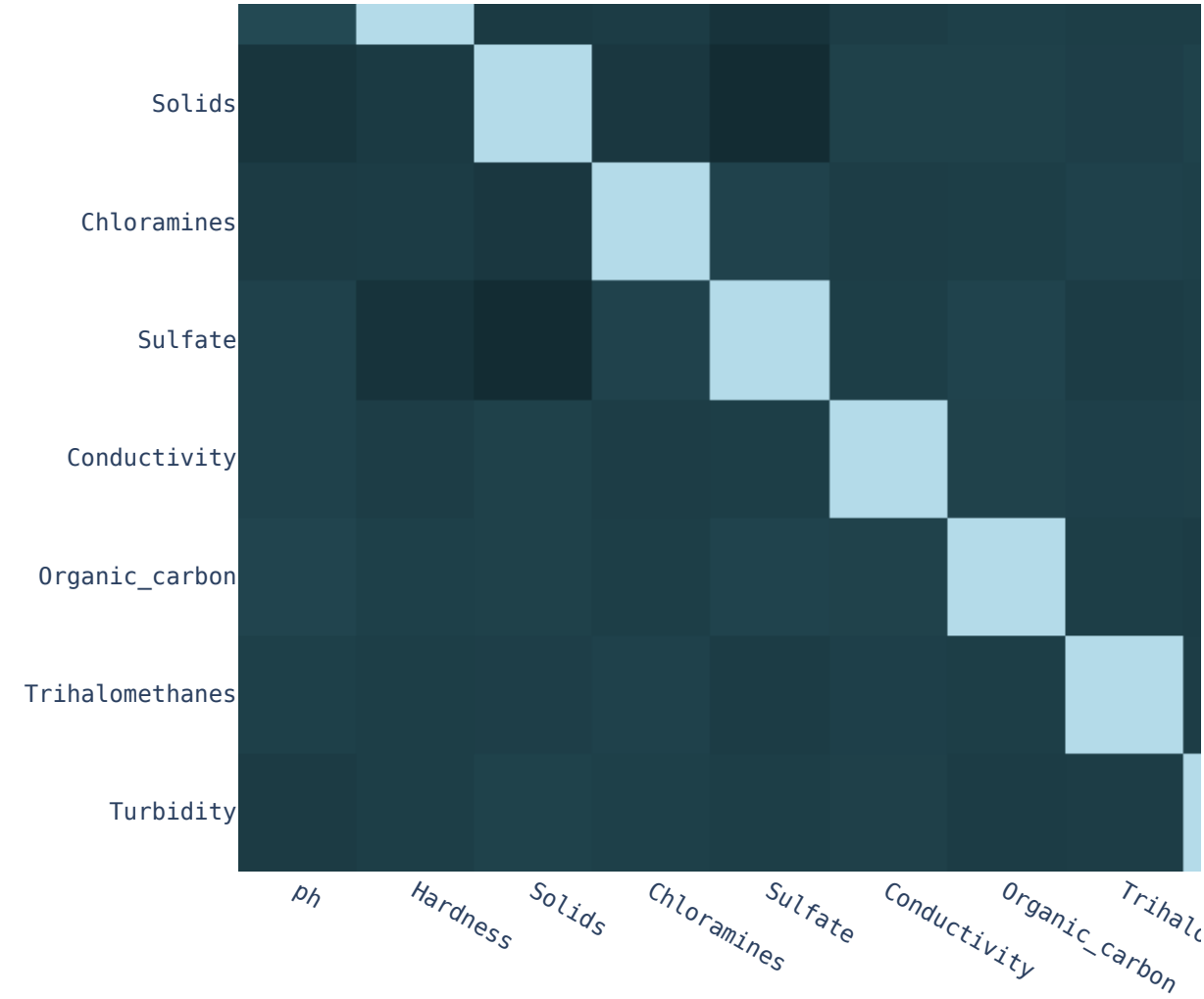
Out[15]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_c
ph	1.000000	0.082096	-0.089288	-0.034350	0.018203	0.018614	0.0
Hardness	0.082096	1.000000	-0.046899	-0.030054	-0.106923	-0.023915	0.0
Solids	-0.089288	-0.046899	1.000000	-0.070148	-0.171804	0.013831	0.0
Chloramines	-0.034350	-0.030054	-0.070148	1.000000	0.027244	-0.020486	-0.0
Sulfate	0.018203	-0.106923	-0.171804	0.027244	1.000000	-0.016121	0.0
Conductivity	0.018614	-0.023915	0.013831	-0.020486	-0.016121	1.000000	0.0
Organic_carbon	0.043503	0.003610	0.010242	-0.012653	0.030831	0.020966	1.0
Trihalomethanes	0.003354	-0.013013	-0.009143	0.017084	-0.030274	0.001285	-0.0
Turbidity	-0.039057	-0.014449	0.019546	0.002363	-0.011187	0.005798	-0.0

Let's make a Heatmap to visualize the correlation.

In [16]:
```
fig = px.imshow(cor,height=800,width=800,color_continuous_scale=colors_blue,t

fig.update_layout(font_family='monospace',
                  title=dict(text='Correlation Heatmap',x=0.5,y=0.93,
                             font=dict(color=colors_dark[2],size=24)),
                  coloraxis_colorbar=dict(len=0.85,x=1.1)
                   )

fig.show()
```





Data Preparation Dealing With Missing Values

In [17]:
```
fig = msno.matrix(df,color=(0,0.5,0.5))
```

We can see that there are many missing values in the dataset, so we'll try to deal with it

Tratamento dos dados

Como podemos ver abaixo existem vários valores `NaN`, *not a number*, na base. Portanto precisaremos tratar os dados, e removendo os valores nulos. Depois realizamos uma técnica conhecida como *undersampling*, na qual igualamos o número de instâncias das duas classes removendo as instâncias da classe com mais valores.

In [18]:
```
df.isnull().sum()
```

Out[18]:
```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

In [19]:
```
df[df['Potability']==0].describe()
```

Out[19]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_
count	1684.000000	1998.000000	1998.000000	1998.000000	1510.000000	1998.000000	1998.
mean	7.085378	196.733292	21777.490788	7.092175	334.564290	426.730454	14.
std	1.683499	31.057540	8543.068788	1.501045	36.745549	80.047317	3.
min	0.000000	98.452931	320.942611	1.683993	203.444521	181.483754	4.
25%	6.037723	177.823265	15663.057382	6.155640	311.264006	368.498530	12.
50%	7.035456	197.123423	20809.618280	7.090334	333.389426	422.229331	14.
75%	8.155510	216.120687	27006.249009	8.066462	356.853897	480.677198	16.
max	14.000000	304.235912	61227.196008	12.653362	460.107069	753.342620	28.

In [20]:
```
df[df['Potability']==1].describe()
```

Out[20]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_c
count	1101.000000	1278.000000	1278.000000	1278.000000	985.000000	1278.000000	1278.0
mean	7.073783	195.800744	22383.991018	7.169338	332.566990	425.383800	14.1
std	1.448048	35.547041	9101.010208	1.702988	47.692818	82.048446	3.2
min	0.227499	47.432000	728.750830	0.352000	129.000000	201.619737	2.2
25%	6.179312	174.330531	15668.985035	6.094134	300.763772	360.939023	12.0
50%	7.036752	196.632907	21199.386614	7.215163	331.838167	420.712729	14.1
75%	7.933068	218.003420	27973.236446	8.199261	365.941346	484.155911	16.3
max	13.175402	323.124000	56488.672413	13.127000	481.030642	695.369528	23.6

In [21]:
```
df[df['Potability']==0][['ph','Sulfate','Trihalomethanes']].median()
```

Out[21]:
```
ph                   7.035456
Sulfate            333.389426
Trihalomethanes     66.542198
dtype: float64
```

We can see that the difference between the mean and median values of Potable and Non-Potable Water is minimal. So we use the overall median of the feature to impute the values

In [22]:
```
df = df.dropna()
df.isnull().sum()
```

Out[22]:
```
ph                 0
Hardness           0
Solids             0
Chloramines        0
Sulfate            0
Conductivity       0
Organic_carbon     0
Trihalomethanes    0
Turbidity          0
Potability         0
dtype: int64
```

In [23]:
```
import numpy as np

not_potable_indices = df[df.Potability == 0].index
number_of_potables = df[df.Potability == 1].shape[0]
random_indices = np.random.choice(not_potable_indices, number_of_potables, re
df_not_potables = df.loc[random_indices]
df_potables = df[df.Potability == 1]
df = pd.concat([df_potables, df_not_potables])

df.Potability.value_counts()
```

Out[23]:
```
0    811
1    811
Name: Potability, dtype: int64
```

Dividindo os dados em treino e teste

Dividimos os dados em X, sendo os instâncias e y os valores da classe "Potability".

In [24]:
```
from sklearn.model_selection import train_test_split

X = df.drop('Potability',axis=1).values
y = df['Potability'].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, rand
```

Treinos e testes

Bernoulli Naive Bayes

In [25]:
```
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import classification_report, accuracy_score
```

```
BNB = BernoulliNB()
BNB.fit(X_train, y_train)

y_predicted = BNB.predict(X_test)

target_names = ['Not Potable', 'Potable']
print(f"Accuracy: {accuracy_score(y_test, y_predicted)}")
print(classification_report(y_test, y_predicted, target_names=target_names))
```

```
Accuracy: 0.4804928131416838
              precision    recall  f1-score   support

 Not Potable       0.48      1.00      0.65       234
     Potable       0.00      0.00      0.00       253

    accuracy                           0.48       487
   macro avg       0.24      0.50      0.32       487
weighted avg       0.23      0.48      0.31       487

/home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/site-pack
ages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

/home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/site-pack
ages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.

/home/aritana/my_jupyter_notebook/my_jupyter_notebook/lib/python3.8/site-pack
ages/sklearn/metrics/_classification.py:1248: UndefinedMetricWarning:

Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
```

Matriz confusao

In [26]:

```
from sklearn.metrics import confusion_matrix
print (confusion_matrix(y_test, y_predicted))
```

```
[[234   0]
 [253   0]]
```

Análise

É interessante notar que o Bernoulli Naive Bayes não conseguiu generalizar bem, classificando todos as instâncias como 0, não potáveis. Aqui nós ainda temos que descobrir o porquê (ou não) do que isso aconteceu pra colocar no artigo. Como não existem praticamente nenhum parâmetro

Random Forest - Conjunto de parâmetros 1

In [27]:

```
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(n_estimators=100, min_samples_leaf=2, random_stat
RF.fit(X_train, y_train)

y_predicted = RF.predict(X_test)
```

```
target_names = ['Not Potable', 'Potable']
print(f"Accuracy: {accuracy_score(y_test, y_predicted)}")
print(classification_report(y_test, y_predicted, target_names=target_names))
```

```
Accuracy: 0.6201232032854209
              precision    recall  f1-score   support

  Not Potable       0.60      0.63      0.62       234
      Potable       0.64      0.61      0.62       253

     accuracy                           0.62       487
    macro avg       0.62      0.62      0.62       487
 weighted avg       0.62      0.62      0.62       487
```

Matriz Confusao

In [31]:

```
from sklearn.metrics import confusion_matrix
print (confusion_matrix(y_test, y_predicted))
```

```
[[159  75]
 [102 151]]
```

Análise

Ao contrário do Bernoulli Naive Bayes, o Random Forest Classifier conseguiu generalizar para as duas classes

Random Forest - Conjunto de parâmetros 2

In [28]:

```
RF = RandomForestClassifier(n_estimators=1000, min_samples_leaf=2, random_sta
RF.fit(X_train, y_train)

y_predicted = RF.predict(X_test)

target_names = ['Not Potable', 'Potable']
print(f"Accuracy: {accuracy_score(y_test, y_predicted)}")
print(classification_report(y_test, y_predicted, target_names=target_names))
```

```
Accuracy: 0.6057494866529775
              precision    recall  f1-score   support

  Not Potable       0.58      0.64      0.61       234
      Potable       0.63      0.58      0.60       253

     accuracy                           0.61       487
    macro avg       0.61      0.61      0.61       487
 weighted avg       0.61      0.61      0.61       487
```

Análise

Como podemos ver, o número de árvores da flores não influencia tanto na melhoria do modelo. Com 10 vezes o número mair de árvores, a melhor no *f1-score* é de apenas 0.01.

Random Forest - Conjunto de parâmetros 3

In [29]:

```
RF = RandomForestClassifier(n_estimators=100, min_samples_leaf=1, random_stat
RF.fit(X_train, y_train)
```

```
y_predicted = RF.predict(X_test)

target_names = ['Not Potable', 'Potable']
print(f"Accuracy: {accuracy_score(y_test, y_predicted)}")
print(classification_report(y_test, y_predicted, target_names=target_names))
```

```
Accuracy: 0.6365503080082136
              precision    recall  f1-score   support

 Not Potable       0.61      0.68      0.64       234
     Potable       0.67      0.60      0.63       253

    accuracy                           0.64       487
   macro avg       0.64      0.64      0.64       487
weighted avg       0.64      0.64      0.64       487
```

Análise

Diminuir o número de instâncias necessárias para que se construa um nó da árvore não constituiu efeito significativo.

In [ ]:

In [ ]: