💧 Water Quality 💧

Dataset Description

- **pH:** pH of water (0 to 14).
- **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.
- **Potability:** Indicates if water is safe for human consumption. Potable -1 and Not potable -0.

Water quality

- refers to the chemical, physical, and biological characteristics of water based on the standards of its usage.It is most frequently used by reference to a set of standards against which compliance, generally achieved through treatment of the water, can be assessed. The most common standards used to monitor and assess water quality convey the health of ecosystems, safety of human contact, extent of water pollution and condition of drinking water. Water quality has a significant impact on water supply and oftentimes determines supply options.

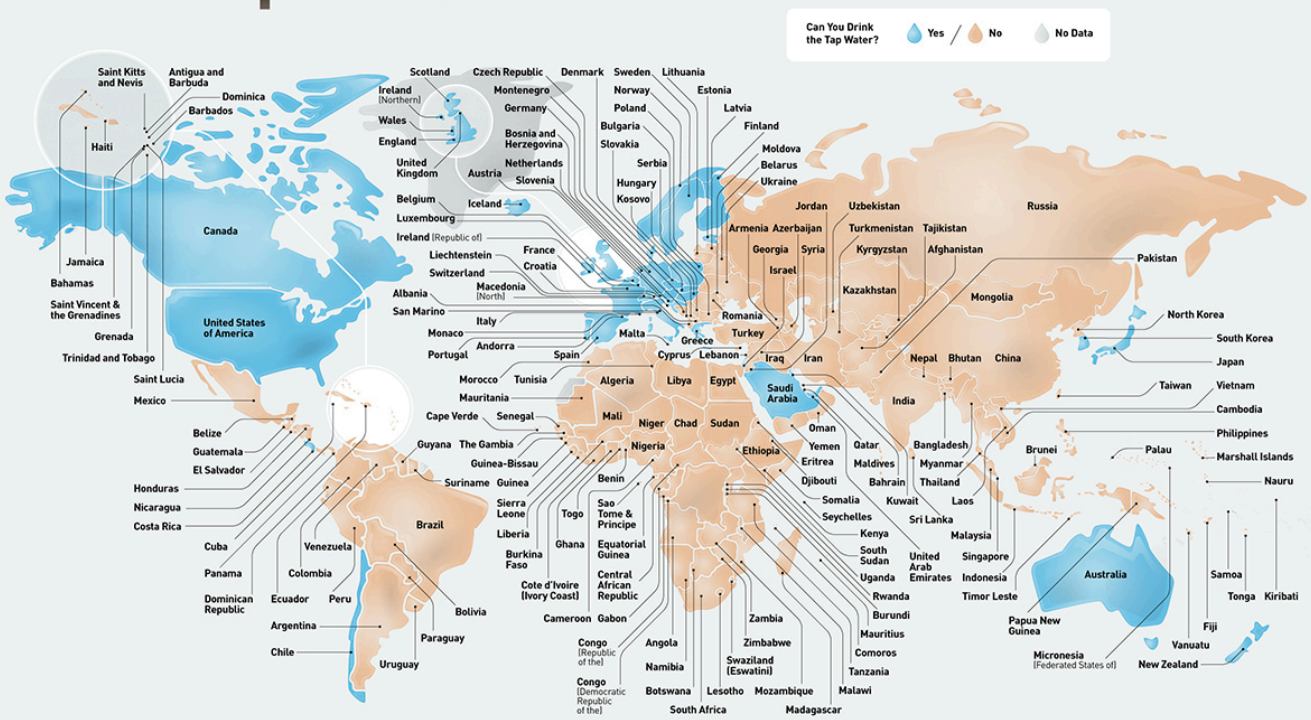
,

THE COUNTRIES WHERE

You Can and Can't Drink Tap Water

While drinkable tap water is taken for granted in some countries, according to the World Health Organization, one in four people around the world doesn't have access to safe drinking water.

We've mapped whether you can safely drink tap water or not in countries around the world, according to CDC advice. Only 50 countries offer drinkable tap water, the majority of which are in Europe. In comparison, only three North American countries (**the U.S.**, **Canada** and **Costa Rica**) and one South American country (**Chile**) have drinkable tap water. Every African country and most countries in parts of Asia and Oceania (including **China** and **the Philippines**) lack safe tap water.



METHODOLOGY: Based on Centers for Disease Control and Prevention (CDC) guidance.

QS

,

The Quality of Drinking Water

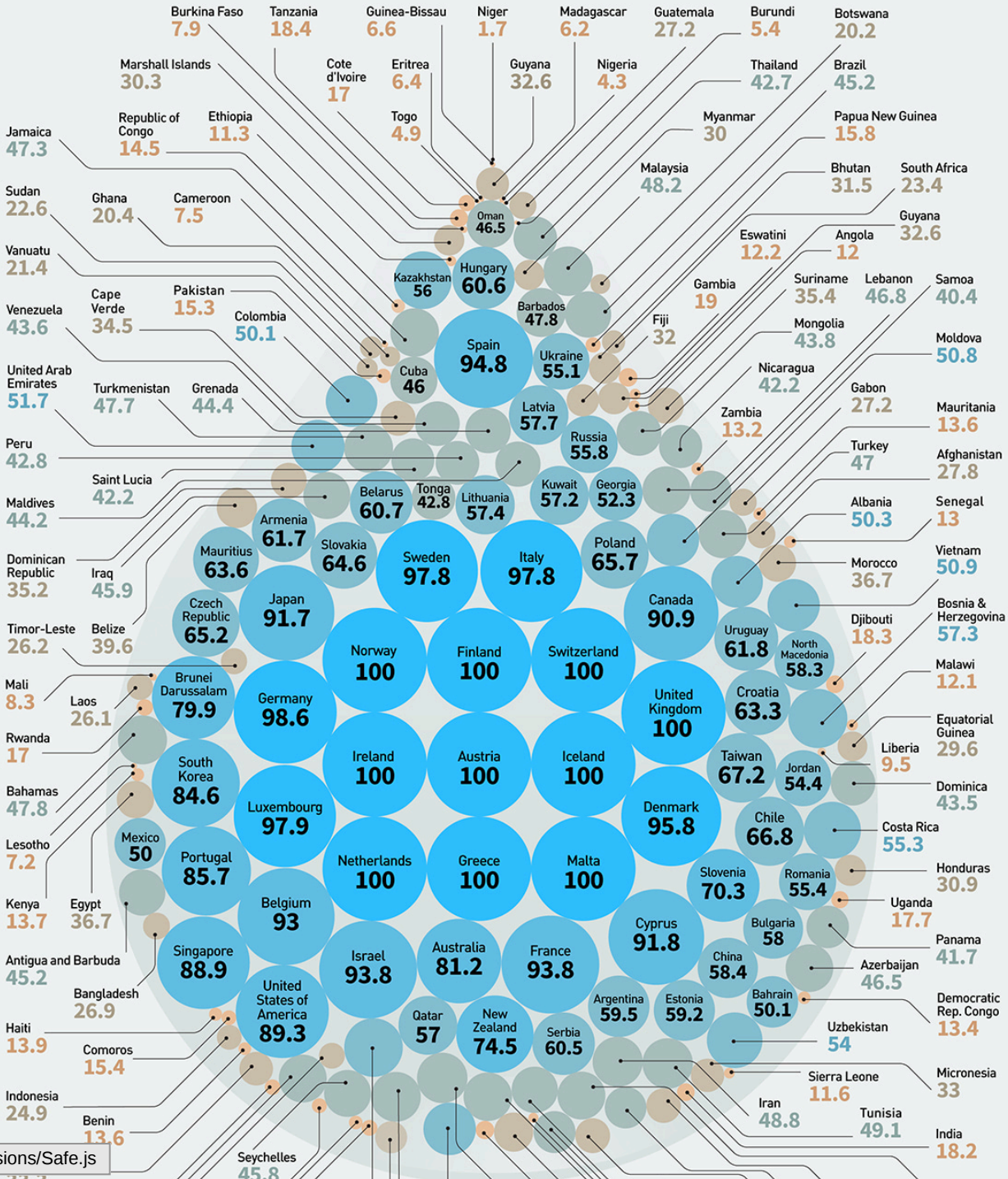
AROUND THE WORLD

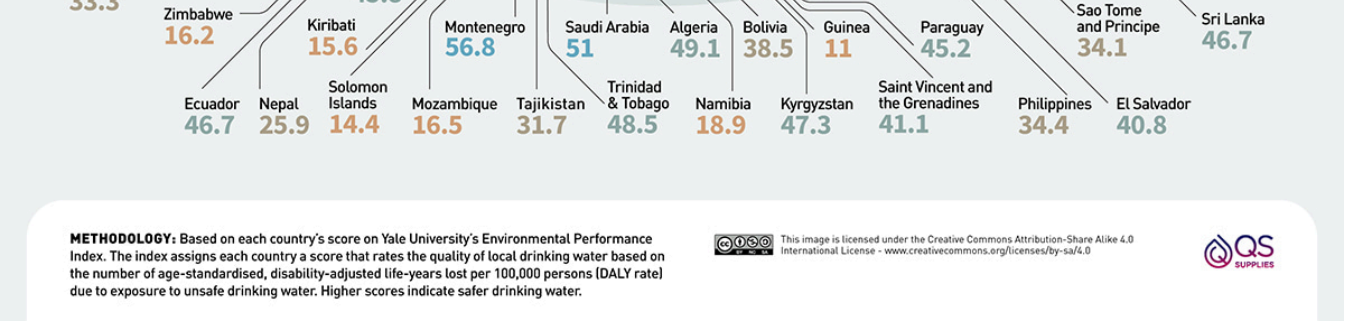
Before water is safe to drink, it first needs to be treated for germs, pollutants and parasites that can make you dangerously unwell. But how does the quality of drinking water differ around the world? Our map reveals each country's water quality (EPI) score as given by Yale University, indicating how safe the local drinking water is. A higher score means safer water.

Ten European countries (including the UK) score a perfect 100, meaning the local tap water is the safest in the world to drink. In contrast, many countries in Africa place at the most dangerous end of the scale.

EPI Score

Least Safe ◄ 0 20 40 50 70 95 100 ► Safest





Importing Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.graph_objects as go
import plotly.express as px
import seaborn as sns
import plotly.subplots as sp
```

Importing Data

In [2]:
```
data = pd.read_csv(r"D:\Elsayed\Projects\Projects\Water Quality\water_potability.csv")
```

In [3]:
```
data
```

Out[3]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethane
0	NaN	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.99097
1	3.716080	129.422921	18630.057858	6.635246	NaN	592.885359	15.180013	56.32907
2	8.099124	224.236259	19909.541732	9.275884	NaN	418.606213	16.868637	66.42009
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.34167
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.99799
...	...	...	...	...	...	...	...	.
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.68769
3272	7.808856	193.553212	17329.802160	8.061362	NaN	392.449580	19.903225	NaN
3273	9.419510	175.762646	33155.578218	7.350233	NaN	432.044783	11.039070	69.84540
3274	5.126763	230.603758	11983.869376	6.303357	NaN	402.883113	11.168946	77.48821
3275	7.874671	195.102299	17404.177061	7.509306	NaN	327.459760	16.140368	78.69844

3276 rows × 10 columns

Exploratory Data Analysis (EDA)

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]:
```
data.isnull().sum()
```

Out[5]:
```
ph                 491
Hardness             0
Solids               0
Chloramines          0
Sulfate            781
Conductivity         0
Organic_carbon       0
Trihalomethanes    162
Turbidity            0
Potability           0
dtype: int64
```

In [6]:
```
print('Percentage(%) of nulls for each columns : ')
pd.DataFrame(data.isnull().sum()*100/len(data))
```

```
Percentage(%) of nulls for each columns :
```

Out[6]:

	0
ph	14.987790
Hardness	0.000000
Solids	0.000000
Chloramines	0.000000
Sulfate	23.840049
Conductivity	0.000000
Organic_carbon	0.000000
Trihalomethanes	4.945055
Turbidity	0.000000
Potability	0.000000

Running this code will generate a detailed HTML report that includes:

- **General Overview**: Provides summary statistics such as the number of rows and columns.
- **Column Statistics**: Displays information for each column, including missing values, unique values, and the most frequent values.
- **Detailed Data Insights**: Features graphical analysis, charts, and relationship analysis between variables.

This report is valuable for understanding the dataset, identifying potential issues such as missing values or outliers, and gaining insights into the data's overall structure.

In [7]:
```
import ydata_profiling
# data.profile_report()
profile = ydata_profiling.ProfileReport(data, title="Pandas Profiling Report", explorative
profile
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Brought to you by YData (https://ydata.ai/?utm_source=opensource&utm_medium=ydataprofiling&utm_campaign=report)

Dataset statistics

Number of variables	10
Number of observations	3276
Missing cells	1434
Missing cells (%)	4.4%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	256.1 KiB
Average record size in memory	80.0 B

Variable types

Numeric	9
Categorical	1

Alerts

`ph` has 491 (15.0%) missing values	**Missing**
`Sulfate` has 781 (23.8%) missing values	**Missing**

Out[7]:

In [8]:
```
# profile.to_notebook_iframe()
profile.to_file("report_profile.html")
```

```
D:\Program Files\anaconda3\Lib\site-packages\ydata_profiling\profile_report.py:363: UserWa
rning: Try running command: 'pip install --upgrade Pillow' to avoid ValueError
  warnings.warn(
Export report to file:   0%|          | 0/1 [00:00<?, ?it/s]
```

- **Quick Overview**: Provides a concise summary of the data's distribution and central tendency.

In [9]:
```
data.describe().T.style.background_gradient(axis=1)
```

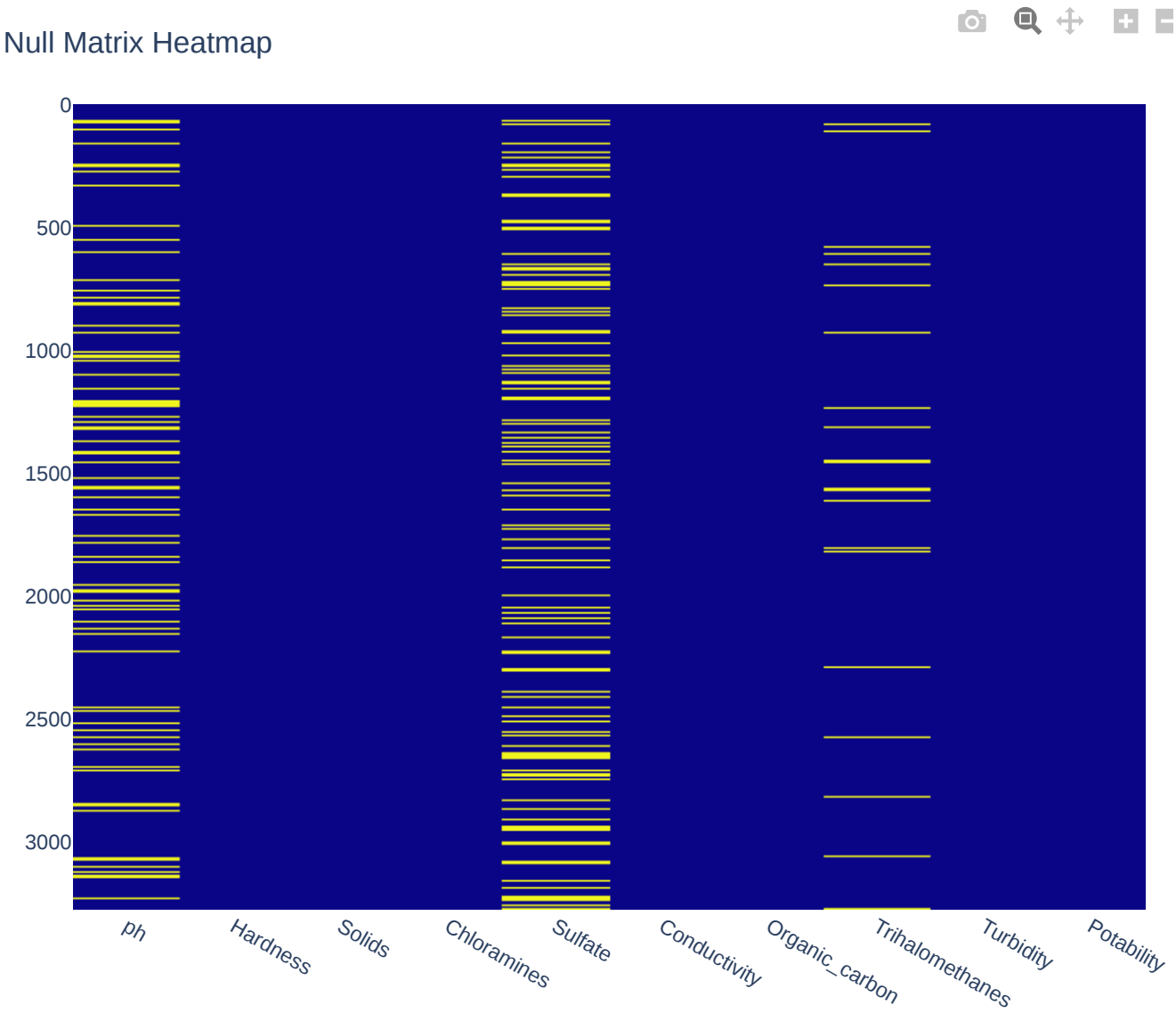
Out[9]:

	count	mean	std	min	25%	50%	75%
ph	2785.000000	7.080795	1.594320	0.000000	6.093092	7.036752	8.062066
Hardness	3276.000000	196.369496	32.879761	47.432000	176.850538	196.967627	216.667456
Solids	3276.000000	22014.092526	8768.570828	320.942611	15666.690297	20927.833607	27332.762127
Chloramines	3276.000000	7.122277	1.583085	0.352000	6.127421	7.130299	8.114887
Sulfate	2495.000000	333.775777	41.416840	129.000000	307.699498	333.073546	359.950170
Conductivity	3276.000000	426.205111	80.824064	181.483754	365.734414	421.884968	481.792304
Organic_carbon	3276.000000	14.284970	3.308162	2.200000	12.065801	14.218338	16.557652
Trihalomethanes	3114.000000	66.396293	16.175008	0.738000	55.844536	66.622485	77.337473
Turbidity	3276.000000	3.966786	0.780382	1.450000	3.439711	3.955028	4.500320
Potability	3276.000000	0.390110	0.487849	0.000000	0.000000	0.000000	1.000000

- **Identifying Missing Data**: The heatmap provides a clear and immediate visual representation of missing data within the dataset, making it easy to identify columns or rows with a high percentage of null values.
- **Pattern Recognition**: By visualizing missing data, patterns such as sequences of missing values or entire missing columns can be quickly spotted.
- **Data Quality Assessment**: Helps in assessing the overall quality of the dataset by highlighting areas that may require data cleaning or imputation before further analysis.

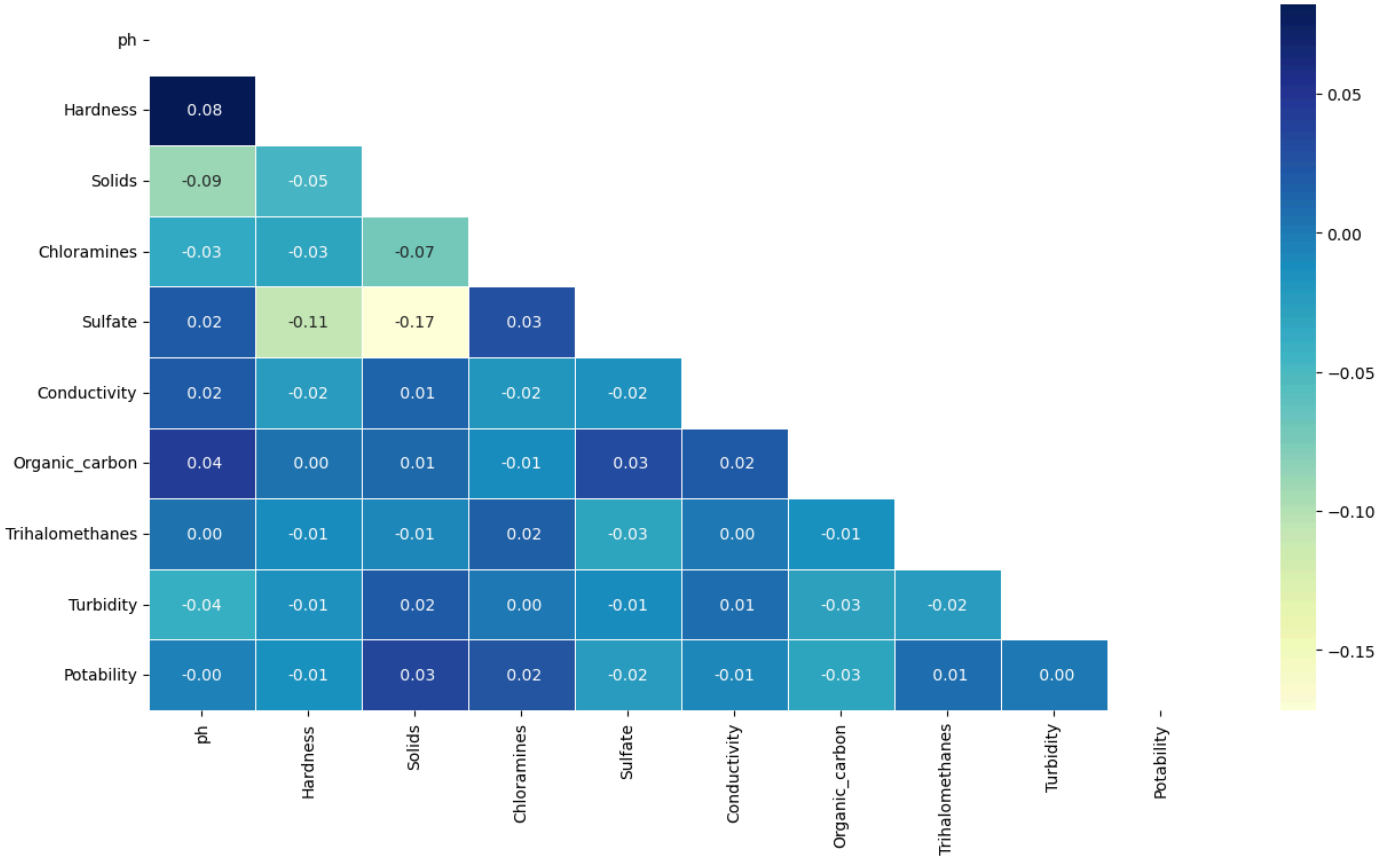
In [10]:

```
correlation_matrix = data.isnull()
fig = px.imshow(correlation_matrix ,
                labels=dict(color="Null Value") ,
                zmin=1 , zmax=1)
fig.update_layout(title="Null Matrix Heatmap" , width=800 , height=600)
fig.show()
```



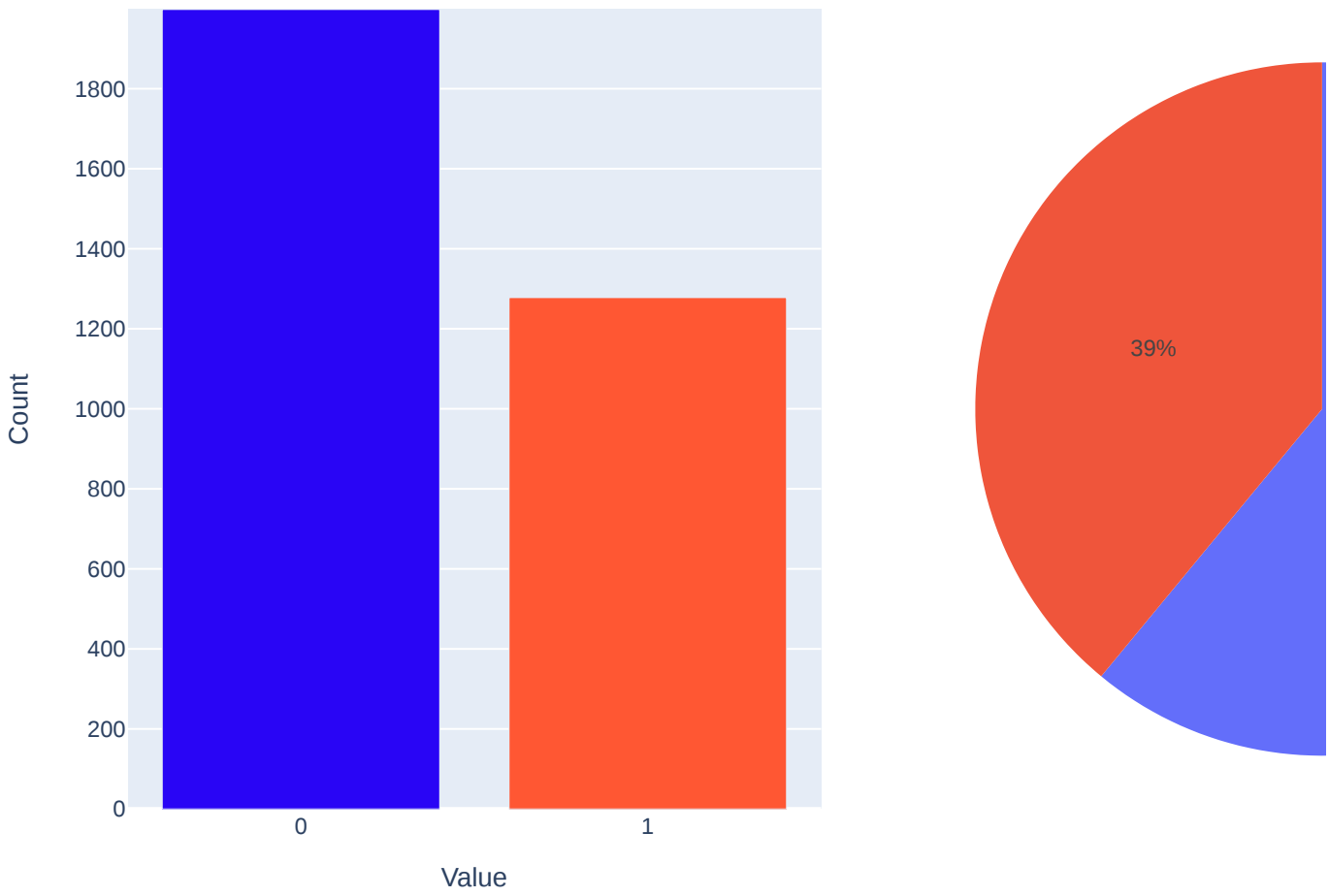
- **Understanding Relationships**: The heatmap helps identify strong positive or negative correlations between numerical variables, which can be critical for feature selection and understanding the relationships within the data.
- **Reducing Multicollinearity**: By visualizing correlations, you can identify highly correlated features, which might cause multicollinearity in predictive models. This insight allows you to consider removing or combining such features.
- **Feature Engineering**: Detecting correlations can inspire the creation of new features or the transformation of existing ones, improving the predictive power of your models.

In [11]:
```
%matplotlib inline

numerical_columns = data.select_dtypes(include=["int64" , "float64"]).columns
numerical_data = data[numerical_columns]
plt.figure(figsize=(15,8))
sns.heatmap(data.corr() , annot=True , fmt=" .2f" , linewidths=0.5 , linecolor="white" , m
plt.show()
```



- **Understanding Class Distribution**: This visualization provides a clear and immediate understanding of how the data is distributed between potable and non-potable categories.
- **Imbalance Detection**: If the dataset is imbalanced (one category dominates), this can inform decisions about data preprocessing steps such as resampling or weighting during model training.
- **Dual Visualization**: The combination of a bar chart and a pie chart allows for a more comprehensive view —while the bar chart shows exact counts, the pie chart illustrates the proportional relationship between the categories.

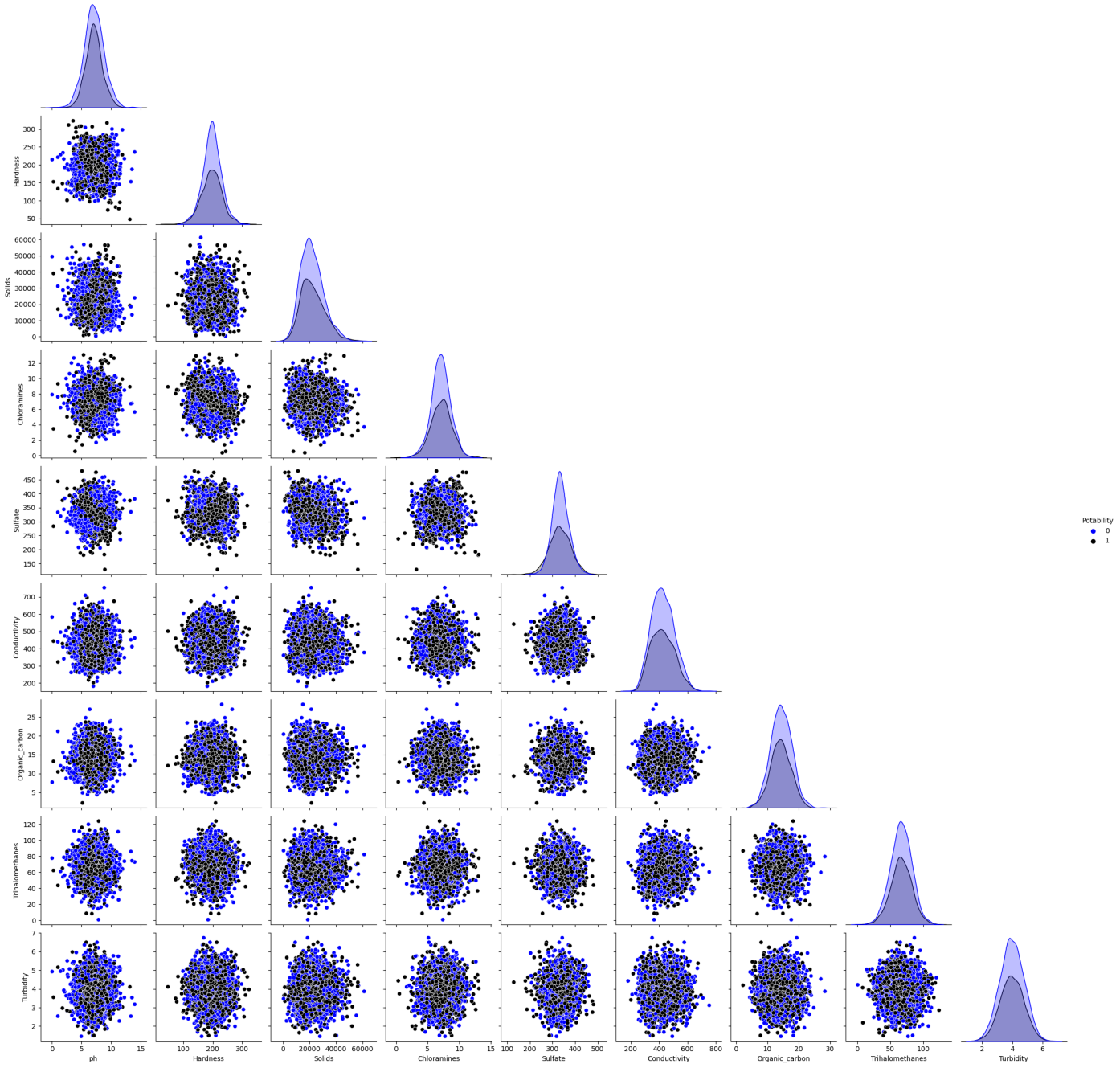
In [12]:
```
potability_count = data["Potability"].value_counts()
fig = sp.make_subplots(rows=1, cols=2, specs=[[{'type':'xy'}, {'type':'domain'}]])
bar_trace = go.Bar(x=potability_count.index, y=potability_count.values , marker=dict(color
fig.add_trace(bar_trace, row=1, col=1)
pie_trace = go.Pie(labels=potability_count.index, values=potability_count.values)
fig.add_trace(pie_trace, row=1, col=2)

fig.update_layout(
    title_text="Potability Distribution: Bar Chart and Pie Chart",
    xaxis_title ="Value",
    yaxis_title="Count" ,
    xaxis=dict(tickmode='linear'),
    yaxis=dict(range=[0, max(potability_count.values) + 1]),
    width=1000,
    height=600
)

fig.show()
```



- **Exploring Feature Relationships**: The pair plot allows you to visualize the relationships and correlations between pairs of features in the dataset. This can help in identifying patterns, trends, or clusters in the data.
- **Class Separation**: By coloring the data points according to the `Potability` feature, you can observe how well the different classes (potable vs. non-potable) are separated across various feature pairs. This is useful for understanding which features may be most predictive of potability.
- **Detecting Multicollinearity**: The pair plot can reveal if certain features are highly correlated with each other, which might lead to multicollinearity issues in predictive modeling.
- **Dimensionality Reduction Insights**: By observing the distribution and overlap of classes, you may gain insights into potential dimensionality reduction techniques or feature engineering strategies that could improve model performance.

In [13]:
```
%matplotlib inline
sns.pairplot(data, hue='Potability', corner=True, palette=['blue', 'black'])
plt.show()
```

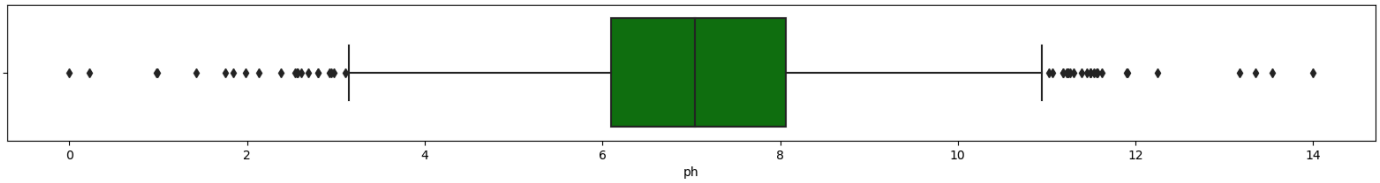
```
D:\Program Files\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning:

The figure layout has changed to tight
```

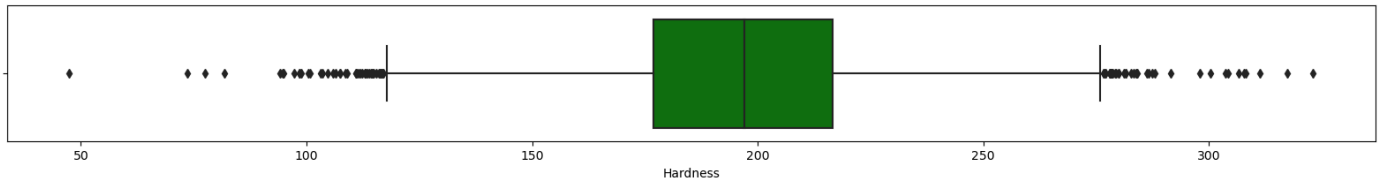


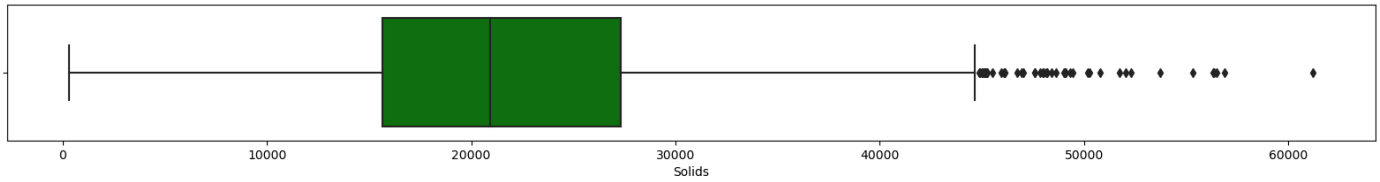
- **Distribution Visualization**: Box plots provide a visual summary of the distribution of data within each column, highlighting the median, quartiles, and potential outliers.
- **Outlier Detection**: The box plots make it easy to identify outliers in the data, which are points that lie outside the whiskers of the box plot. This is crucial for understanding the variability in the data and for deciding on potential data cleaning steps.
- **Comparative Analysis**: By visualizing multiple box plots together, you can compare the distributions of different variables side by side. This helps in identifying which features have a similar range, variance, or presence of outliers.
- **Data Normality**: The shape of the box plots can give an indication of whether the data is symmetrically distributed, skewed, or contains heavy tails, which is important for choosing appropriate statistical methods or transformations.

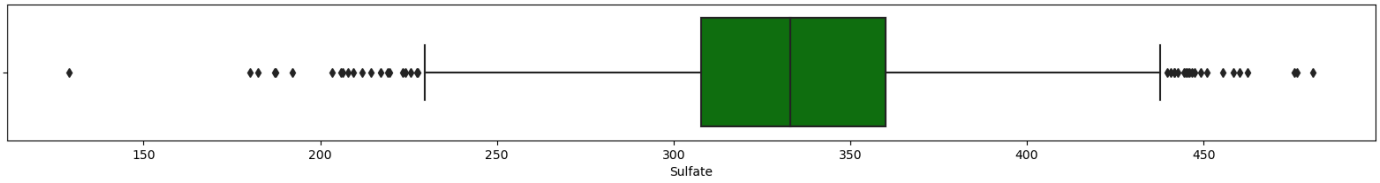
In [14]:
```
%matplotlib inline

column_names=data.columns
fig,axis=plt.subplots(9,1,figsize=(20,30))
fig.subplots_adjust(hspace=0.75)
```

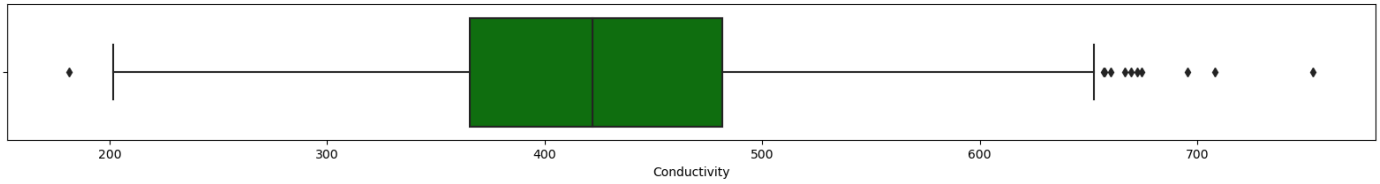
```
for i in range(9):
    sns.boxplot(x=column_names[i],data=data,ax=axis[i] , color="green")
```

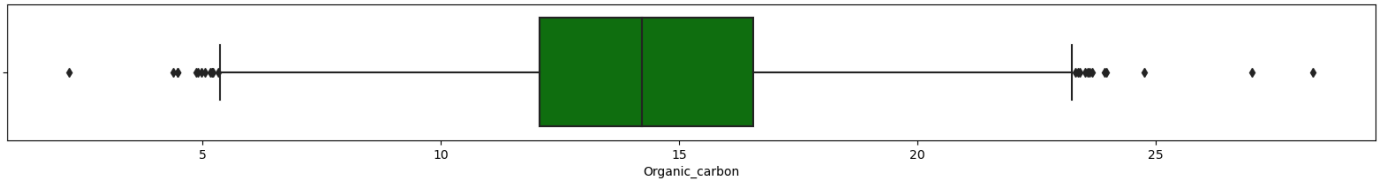


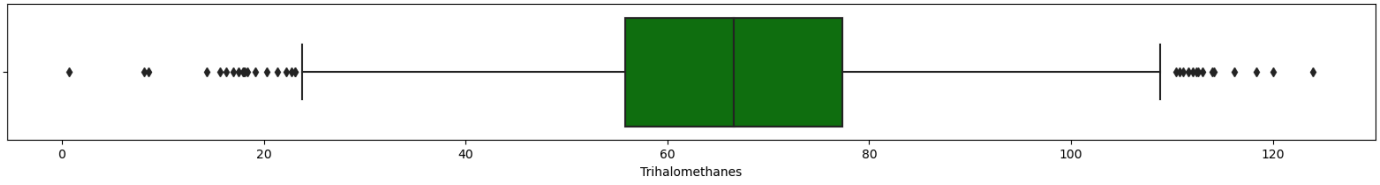


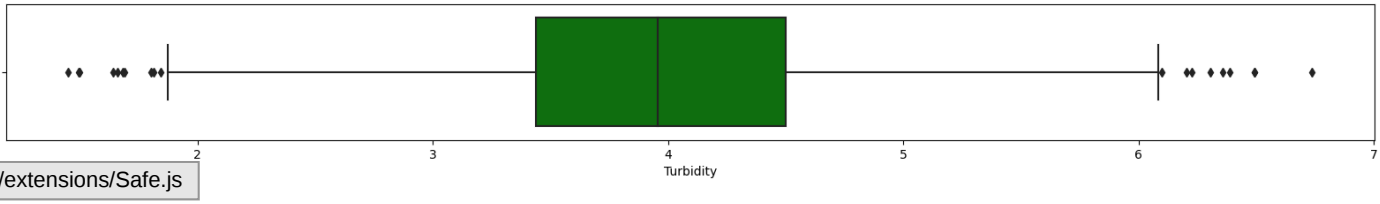






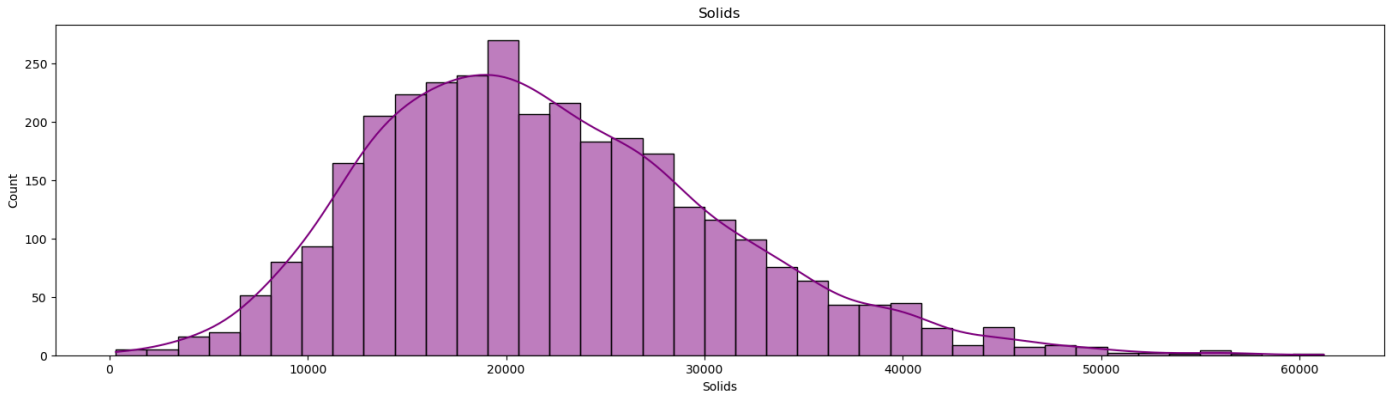


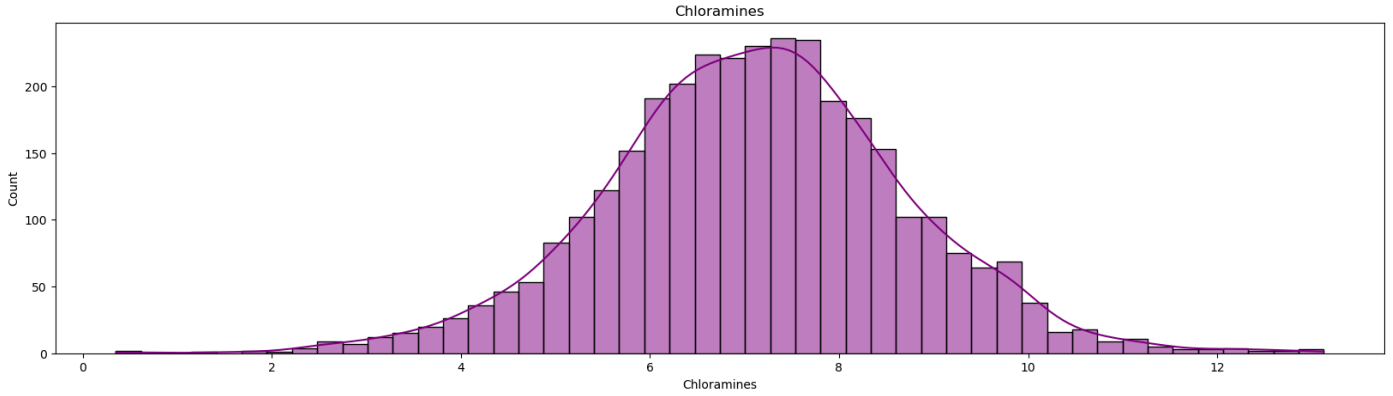


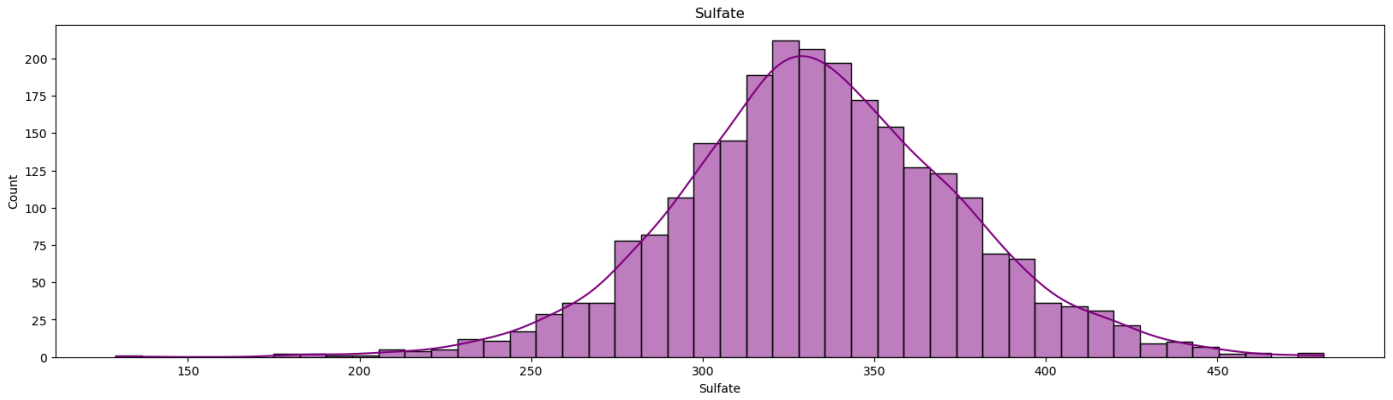


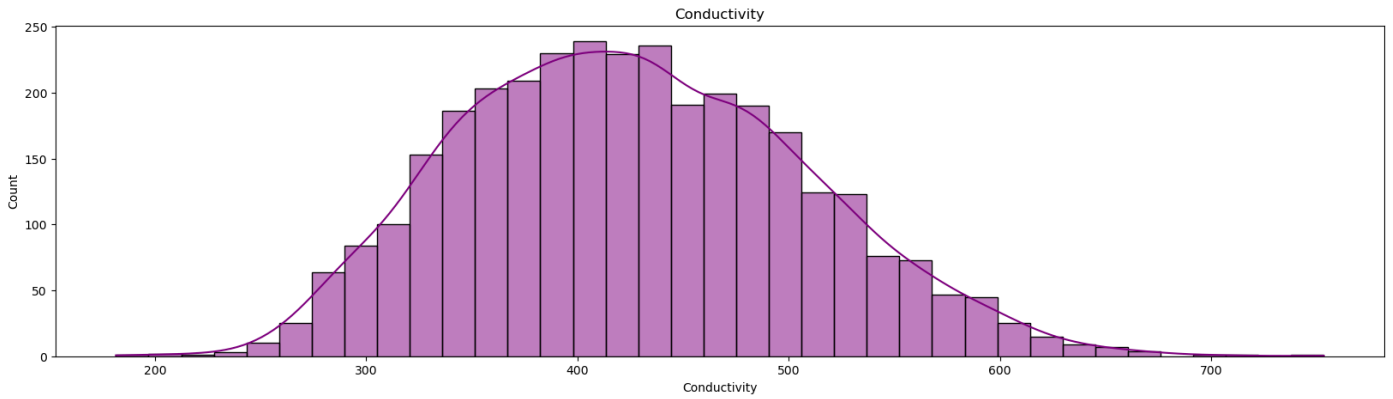
- **Distribution Visualization**: Histograms provide a clear view of the distribution of data within each column, allowing you to see how frequently different values occur.
- **Density Estimation**: The KDE plot superimposed on the histogram offers a smooth approximation of the data distribution, making it easier to identify the underlying distribution shape without the granularity of the histogram bins.
- **Feature-Specific Insights**: By generating a separate plot for each feature, you can gain insights into the individual characteristics of each variable in the dataset, such as whether the data is skewed, bimodal, or normally distributed.
- **Data Skewness and Symmetry**: The combined histogram and KDE plot allow you to quickly assess the skewness and symmetry of the data distribution, which is essential for deciding on appropriate data transformations or modeling techniques.
- **Visualization of All Features**: The loop iterates over all columns, ensuring that the distribution of every feature in the dataset is visualized. This is particularly useful for datasets with many variables, as it provides a comprehensive view of the data.

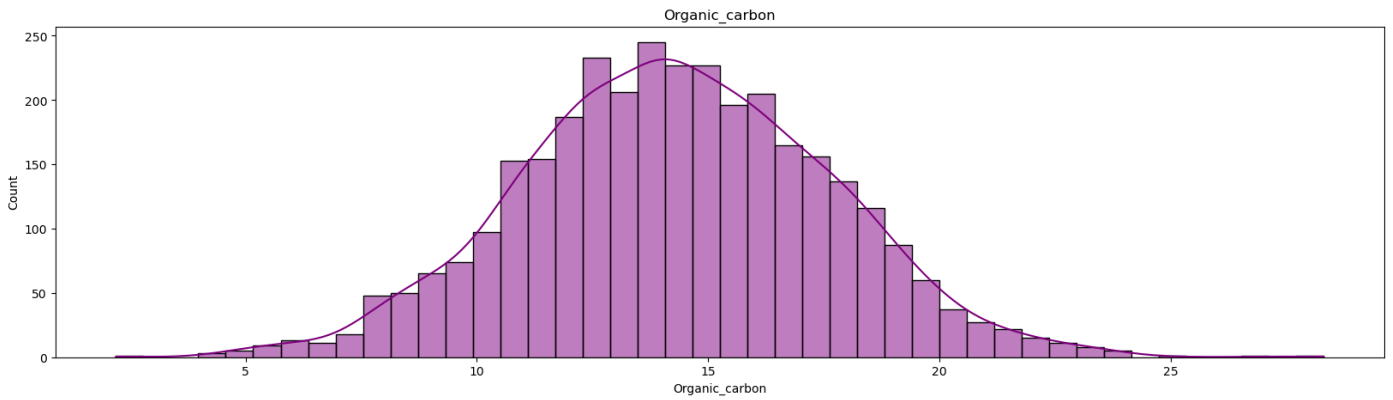
In [15]:
```
%matplotlib inline

for col in data.columns:
        plt.figure(figsize=(20 , 5))
        sns.histplot(x=data[col],kde=True,color='purple')
        plt.title(col)
        plt.show()
```

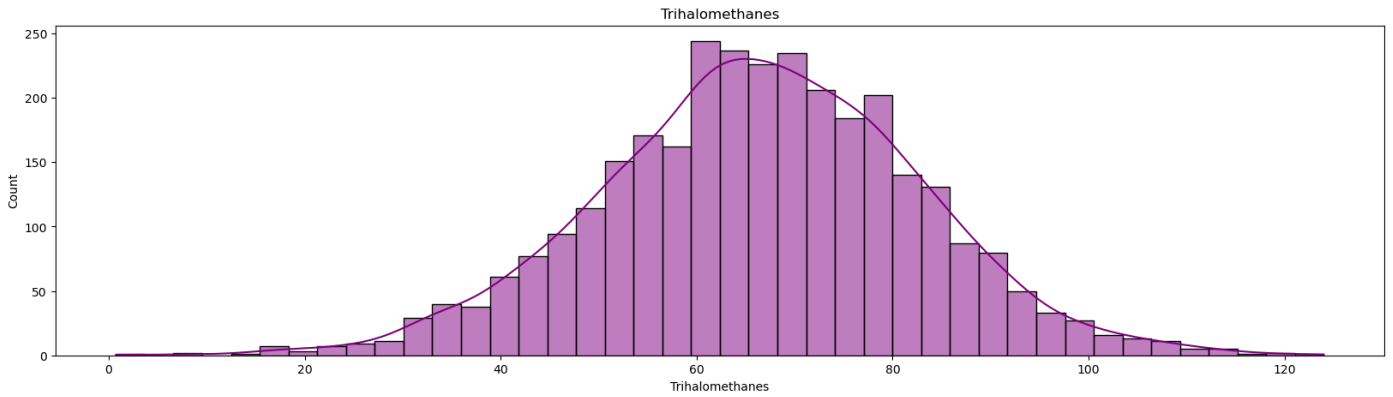


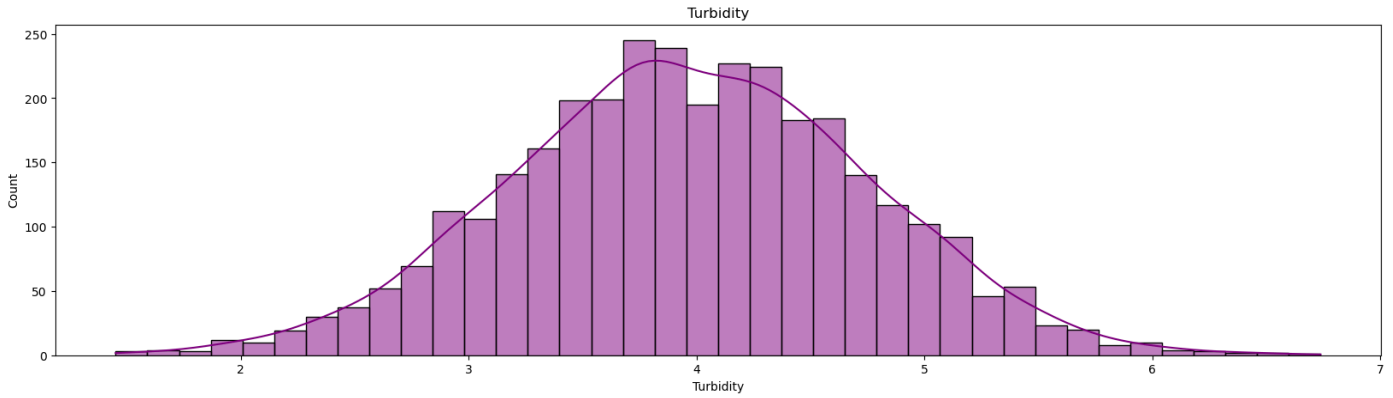


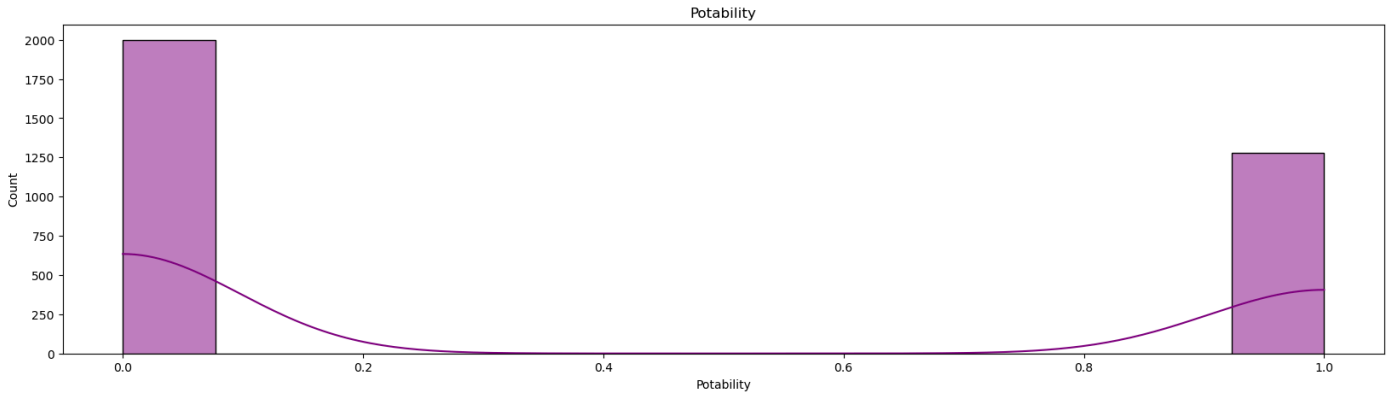












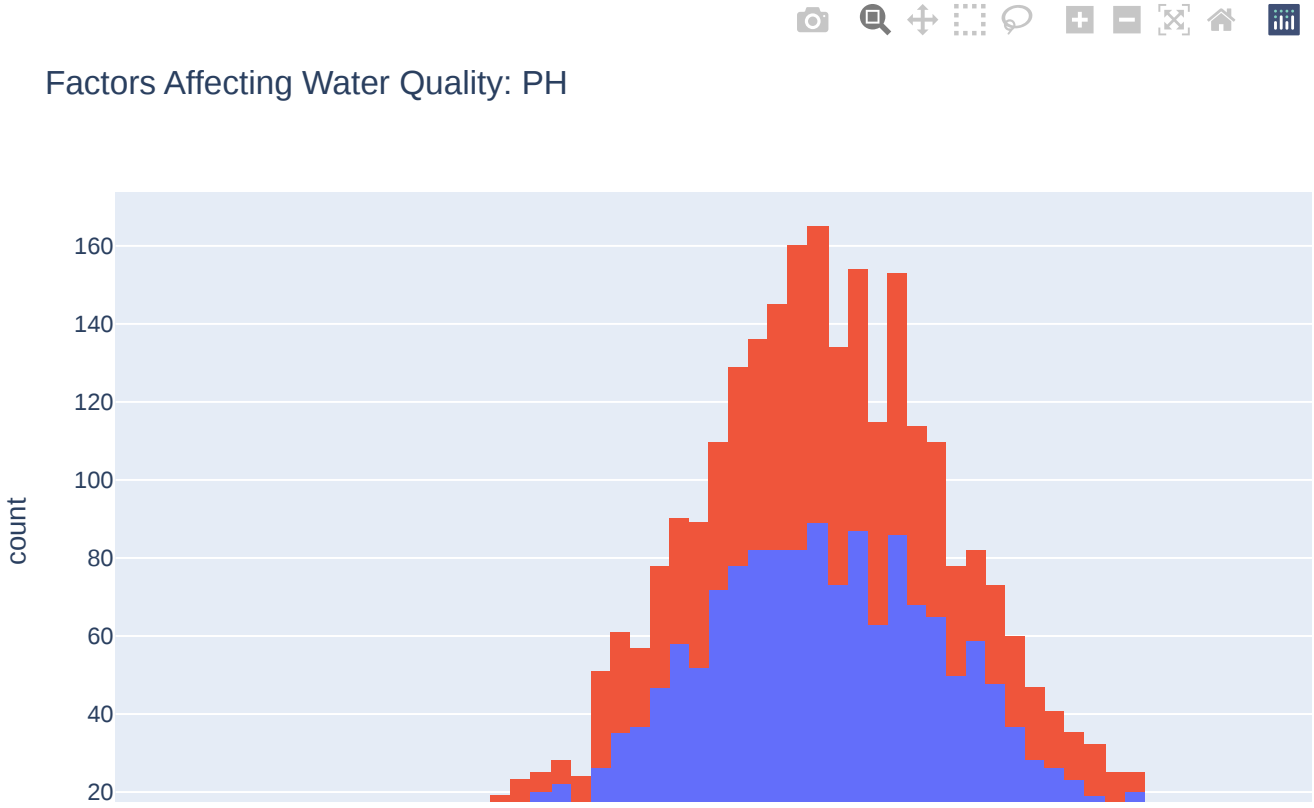
1: PH Column □□

pH Levels and Their Explanations

pH Range	Explanation
0 - 3	Highly acidic, can be corrosive and dangerous.
4 - 6	Moderately acidic, typically found in soft drinks and acidic foods.
7	Neutral pH, like pure water.
8 - 10	Moderately basic, found in many cleaning products and seawater.
11 - 14	Highly basic, can be caustic and used in strong cleaning agents.

- **The ph column represents the ph value of the water which is an important factor in evaluating the acid-base balance of the water. The pH value of drinking water should be between 6.5 and 8.5.**

In [16]:
```
figure = px.histogram(data, x = "ph",
                      color = "Potability",
                      title= "Factors Affecting Water Quality: PH")
figure.show()
```

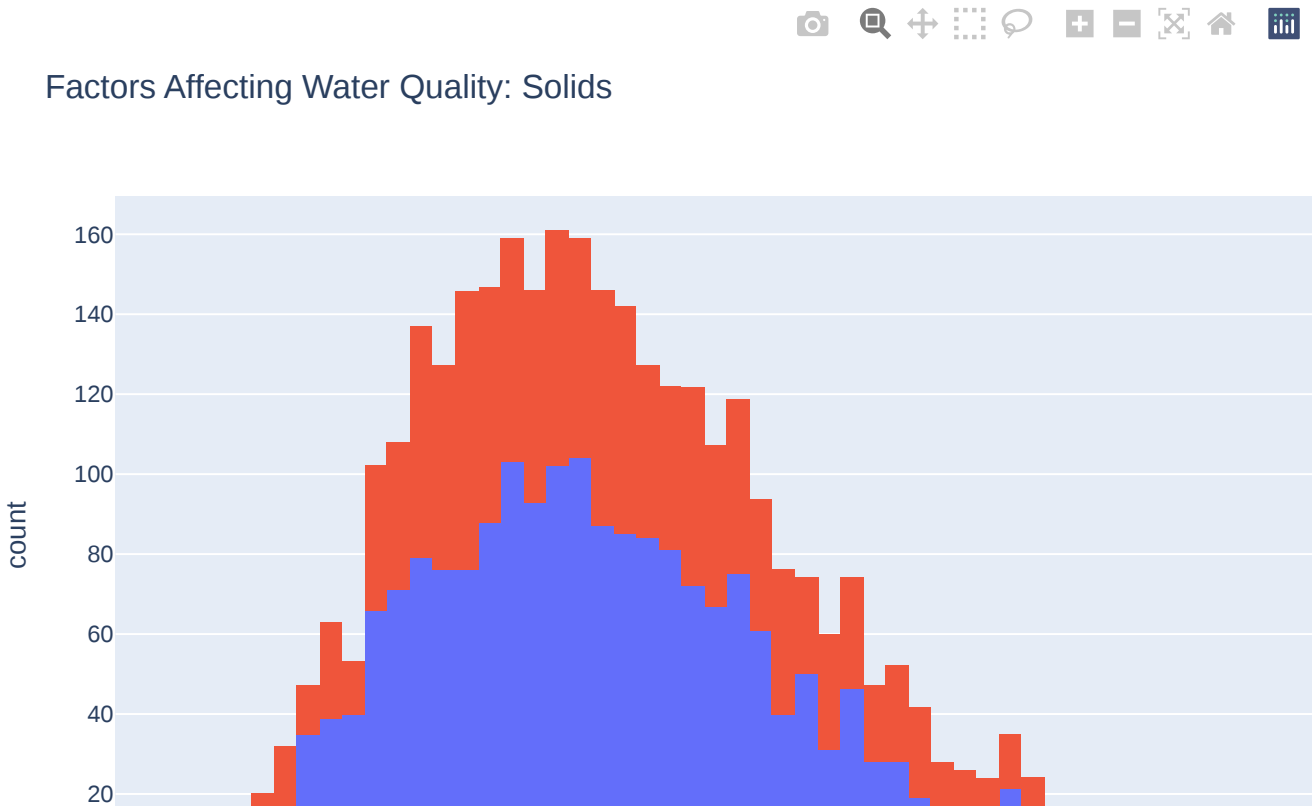


2: Water Hardness

Water Hardness Classification

- **The figure shows the distribution of water hardness in the dataset. The hardness of water usually depends on its source, but water with a hardness of 120-200 milligrams is drinkable.**

In [17]:
```
figure=px.histogram(data,x='Solids',color='Potability',title="Factors Affecting Water Qual
figure.show()
```



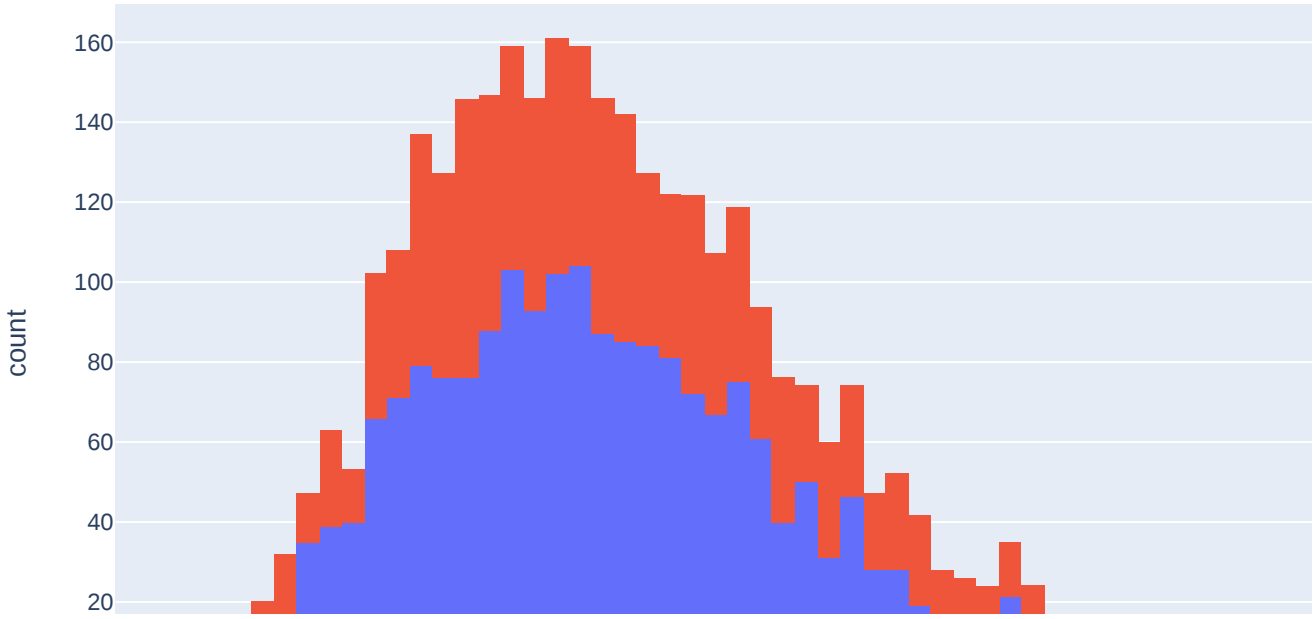
3: Solids

Water Quality Dataset - Solids Explanation

Column Name	Description
Solids	The total amount of dissolved solids in water, including minerals and organic matter. This is an important indicator of water quality.

- **The figure represents the distribution of total dissolved solids in water in the dataset. All organic and inorganic minerals present in water are called dissolved solids. Water with a very high number of dissolved solids is highly mineralized.**

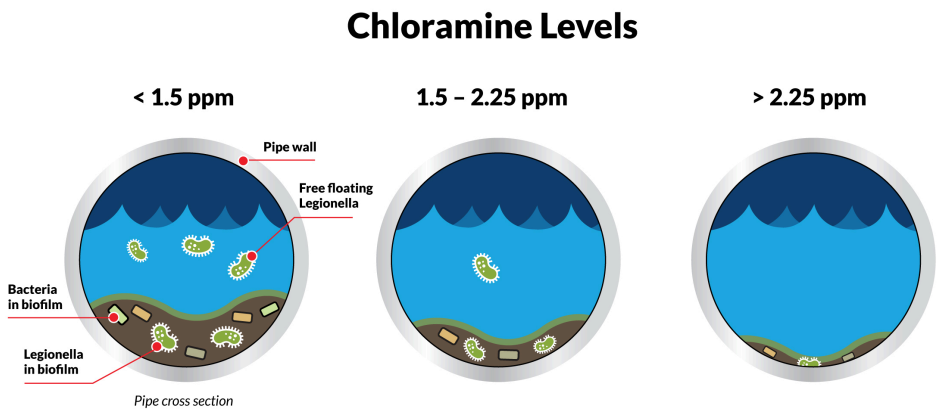
In [18]:
```
figure=px.histogram(data,x='Solids',color='Potability',title="Factors Affecting Water Qual
figure.show()
```



4: Chloramines

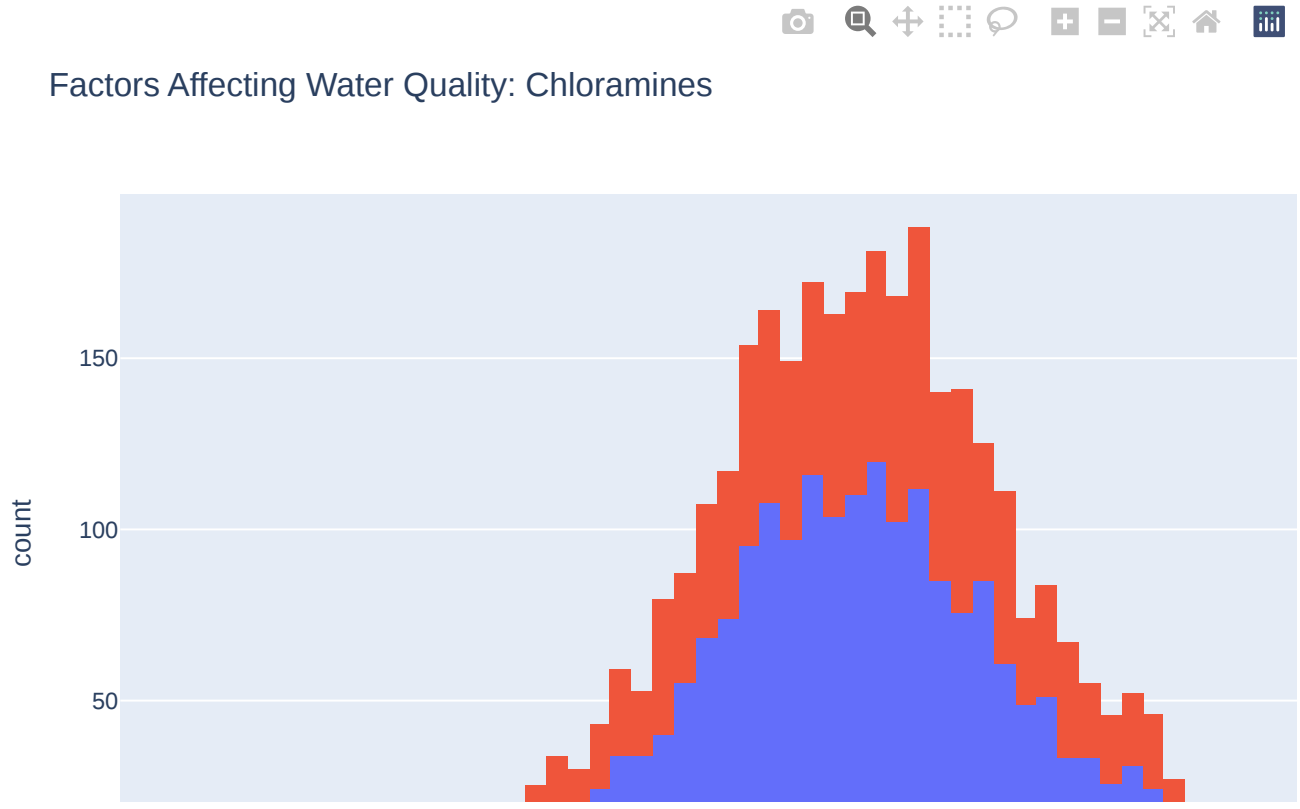
Chloramines Levels and Their Explanations

Chloramines are used as a disinfectant in water treatment. The levels of chloramines can affect water quality and taste. The image below illustrates different levels of chloramines and their implications:



- **The figure represents the distribution of chloramine in water in the dataset. Chloramine and chlorine are disinfectants used in public water systems.**

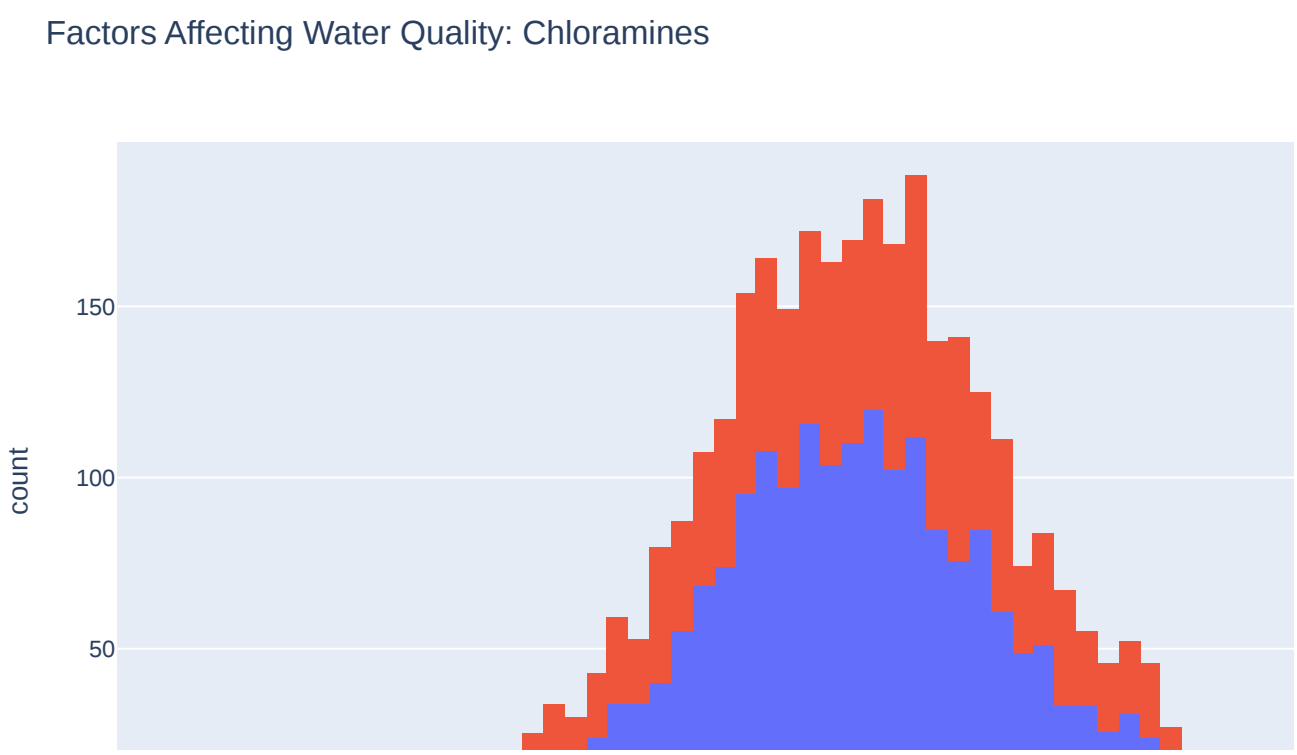
In [19]:
```
figure = px.histogram(data, x = "Chloramines",
                      color = "Potability",
                      title= "Factors Affecting Water Quality: Chloramines")
figure.show()
```



5: Sulfate

- **The figure shows the distribution of sulfate in water in the dataset. They are substances naturally present in minerals, soil and rocks. Water containing less than 500 milligrams of sulfate is safe to drink.**

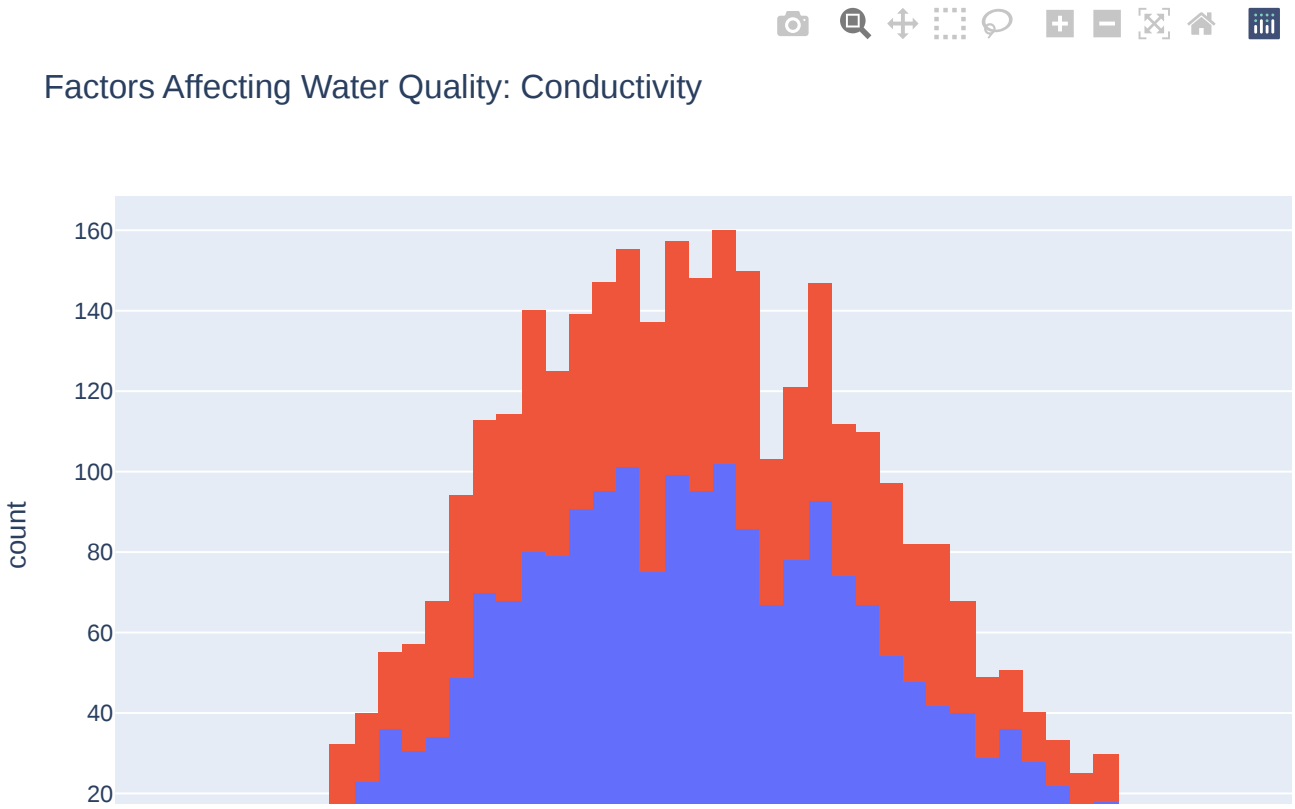
In [20]:
```
igure = px.histogram(data, x = "Sulfate",
                      color = "Potability",
                      title= "Factors Affecting Water Quality: Sulfate")
figure.show()
```



6: Conductivity

- **The figure represents the distribution of water conductivity in the dataset. Water is a good conductor of electricity, but the purest form of water is not a good conductor of electricity. Water with an electrical conductivity of less than 500 is drinkable.**

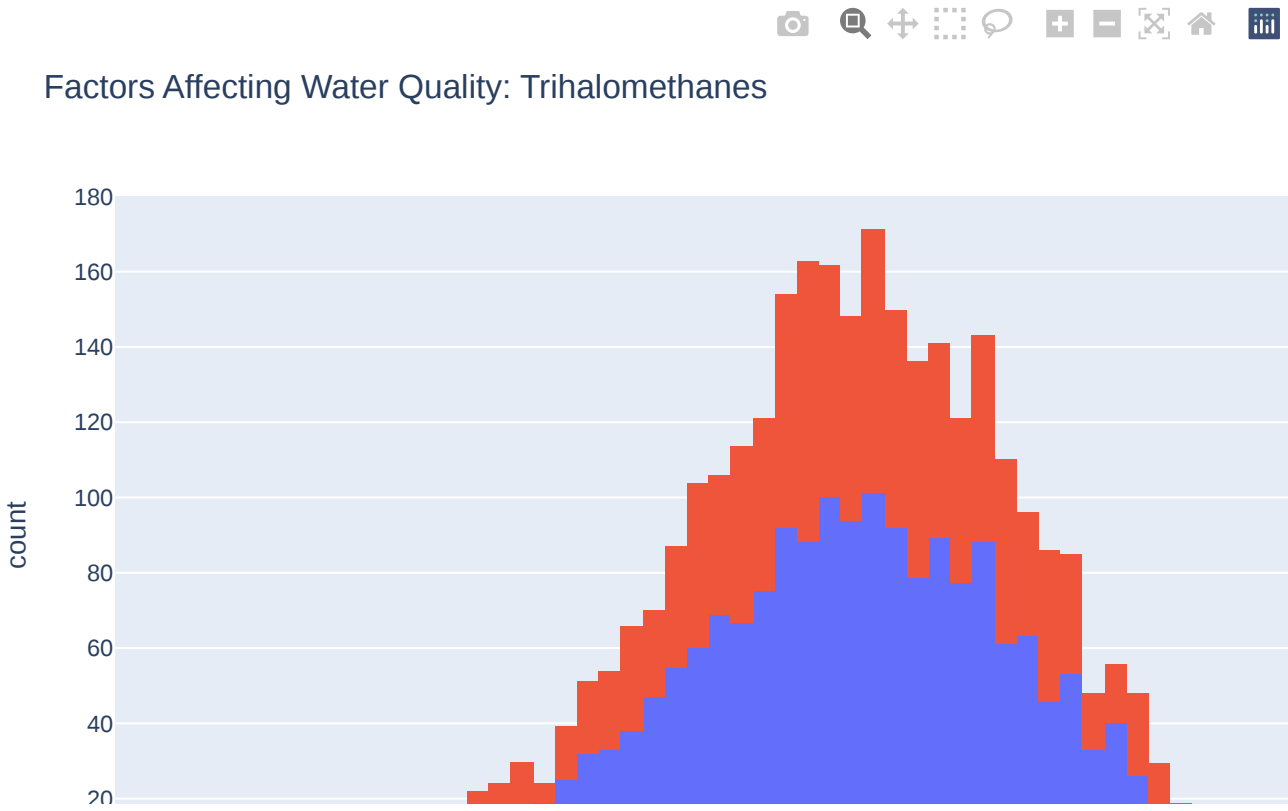
In [21]:
```
figure = px.histogram(data, x = "Conductivity",
                      color = "Potability",
                      title= "Factors Affecting Water Quality: Conductivity")
figure.show()
```



7: Trihalomethanes

- **The figure represents the distribution of trihalomethanes or THMs in water in the dataset. THMs are chemicals found in chlorine-treated water. Water containing less than 80 milligrams of THMs is considered safe to drink.**

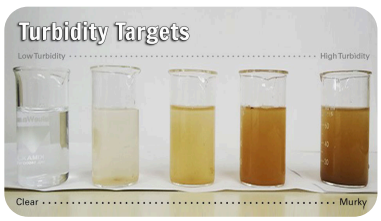
In [22]:
```
figure = px.histogram(data, x = "Trihalomethanes",
                      color = "Potability",
                      title= "Factors Affecting Water Quality: Trihalomethanes")
figure.show()
```



8: Turbidity

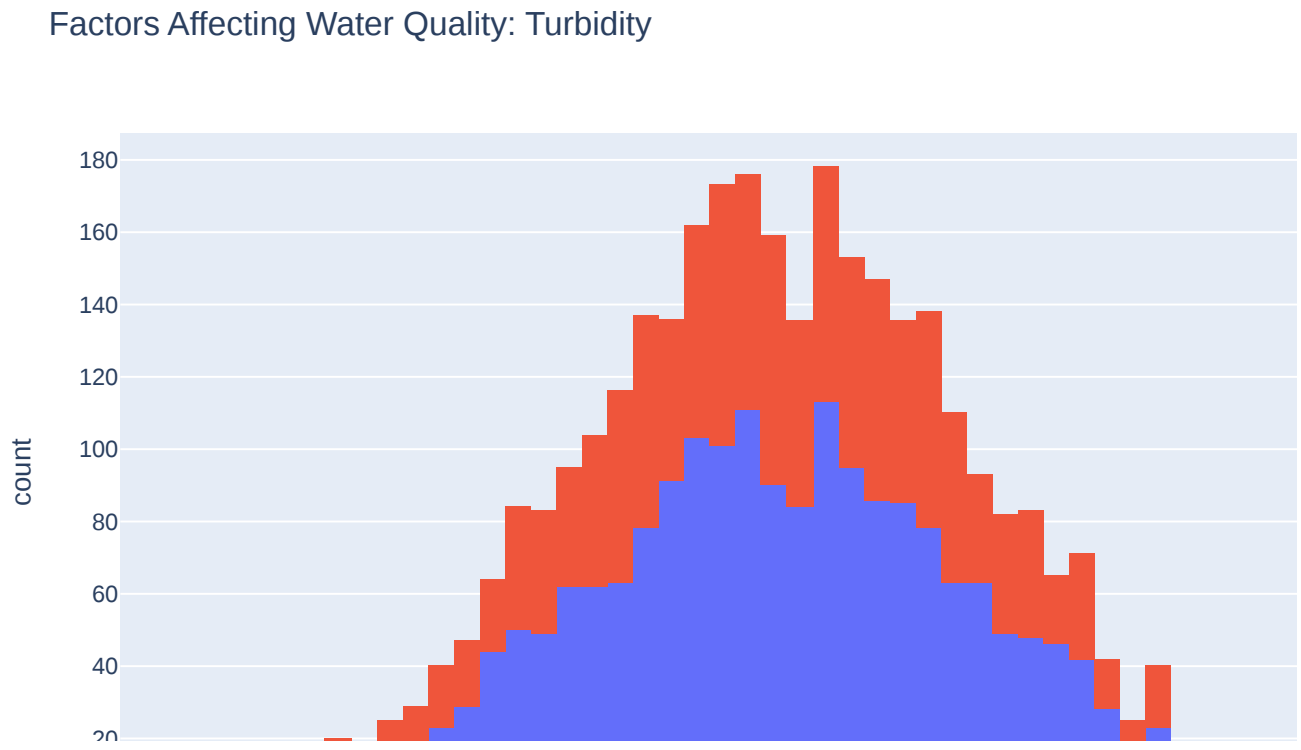
Turbidity

Turbidity is a measure of the clarity of water. It indicates the presence of suspended particles that can affect water quality. The images below illustrate turbidity tests and examples of turbidity in water:



Turbidity is important because it can indicate the presence of microbiological contaminants or organic and inorganic matter that can affect water potability. High turbidity levels can also interfere with water disinfection treatments, making the process less effective.

In [23]:
```
figure = px.histogram(data, x = "Turbidity",
                      color = "Potability",
                      title= "Factors Affecting Water Quality: Turbidity")
figure.show()
```



Data Pre-Processing

In [24]:
```
pd.DataFrame(data.isnull().sum())
```

Out[24]:

	0
ph	491
Hardness	0
Solids	0
Chloramines	0
Sulfate	781
Conductivity	0
Organic_carbon	0
Trihalomethanes	162
Turbidity	0
Potability	0

In [25]:
```
print('Percentage(%) of nulls for each columns : ')
pd.DataFrame(data.isnull().sum()*100/len(data))
```

```
Percentage(%) of nulls for each columns :
```

Out[25]:

	0
ph	14.987790
Hardness	0.000000
Solids	0.000000
Chloramines	0.000000
Sulfate	23.840049
Conductivity	0.000000
Organic_carbon	0.000000
Trihalomethanes	4.945055
Turbidity	0.000000
Potability	0.000000

- **➡ Because of larg number of missing values, we replace them with median for each column.**
  columns with missing values : - ph - Sulfate - Trihalomethanes

In [26]:
```
print("For Potability=1")
data[data.Potability==1][['ph','Sulfate','Trihalomethanes']].median()
```

Out[26]:
```
For Potability=1
ph                   7.036752
Sulfate            331.838167
Trihalomethanes     66.678214
dtype: float64
```

In [27]:
```
print("For Potability=0")
data[data.Potability==0][['ph','Sulfate','Trihalomethanes']].median()
```

Out[27]:
```
For Potability=0
ph                   7.035456
Sulfate            333.389426
Trihalomethanes     66.542198
dtype: float64
```

- **➡ For both Potability=1 and Potability=0, medians are approximately equall.**
- **➡Replace them all with their meadians.**

In [28]:
```
data['ph']=data['ph'].fillna(value=data['ph'].median())
data['Sulfate'] = data['Sulfate'].fillna(value=data['Sulfate'].median())
data['Trihalomethanes'] = data['Trihalomethanes'].fillna(value=data['Trihalomethanes'].me
```

In [29]:
```
print("The Null Value =" , data.isna().sum().sum() )
```

```
The Null Value = 0
```

In [30]:
```
x = data.drop(["Potability"] ,axis=1)
y = data["Potability"]
```

In [31]:
```
x
```

Out[31]:

	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethan
0	7.036752	204.890455	20791.318981	7.300212	368.516441	564.308654	10.379783	86.9909
1	3.716080	129.422921	18630.057858	6.635246	333.073546	592.885359	15.180013	56.3290
2	8.099124	224.236259	19909.541732	9.275884	333.073546	418.606213	16.868637	66.4200
3	8.316766	214.373394	22018.417441	8.059332	356.886136	363.266516	18.436524	100.3416
4	9.092223	181.101509	17978.986339	6.546600	310.135738	398.410813	11.558279	31.9979
...	...	...	...	...	...	...	...	
3271	4.668102	193.681735	47580.991603	7.166639	359.948574	526.424171	13.894419	66.6876
3272	7.808856	193.553212	17329.802160	8.061362	333.073546	392.449580	19.903225	66.6224
3273	9.419510	175.762646	33155.578218	7.350233	333.073546	432.044783	11.039070	69.8454
3274	5.126763	230.603758	11983.869376	6.303357	333.073546	402.883113	11.168946	77.4882
3275	7.874671	195.102299	17404.177061	7.509306	333.073546	327.459760	16.140368	78.6984

3276 rows × 9 columns

- **➡ Normalization**

In [32]:
```
from sklearn.preprocessing import MinMaxScaler
minmaxscaler = MinMaxScaler()
x = pd.DataFrame(minmaxscaler.fit_transform(x) , columns=x.columns)
```

In [33]:
```
x
```

Out[33]:

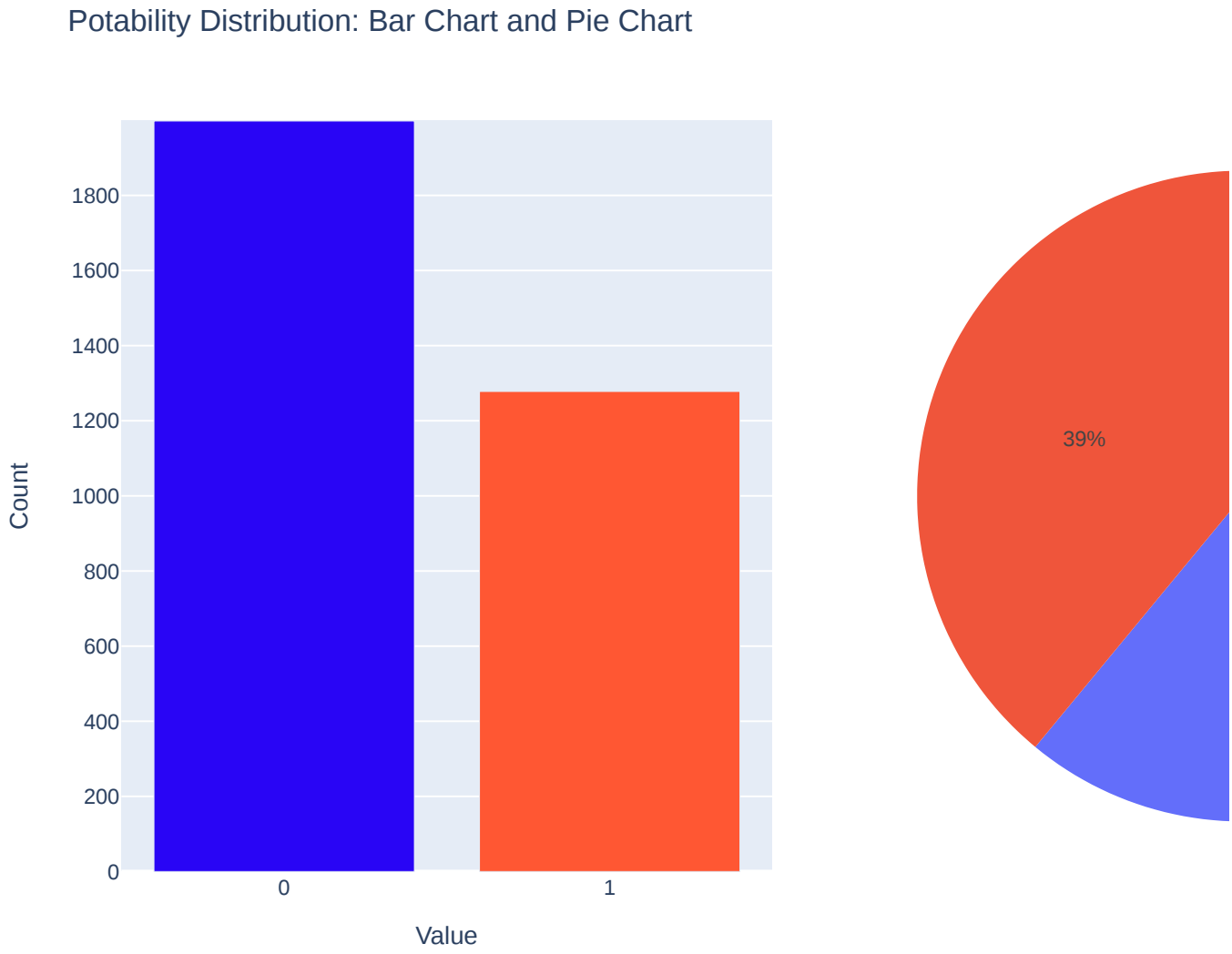
	ph	Hardness	Solids	Chloramines	Sulfate	Conductivity	Organic_carbon	Trihalomethanes	Turl
0	0.502625	0.571139	0.336096	0.543891	0.680385	0.669439	0.313402	0.699753	0.28
1	0.265434	0.297400	0.300611	0.491839	0.579704	0.719411	0.497319	0.450999	0.57
2	0.578509	0.641311	0.321619	0.698543	0.579704	0.414652	0.562017	0.532866	0.30
3	0.594055	0.605536	0.356244	0.603314	0.647347	0.317880	0.622089	0.808065	0.60
4	0.649445	0.484851	0.289922	0.484900	0.514545	0.379337	0.358555	0.253606	0.49
...	...	...	...	...	...	...	...	...	
3271	0.333436	0.530482	0.775947	0.533436	0.656047	0.603192	0.448062	0.535037	0.56
3272	0.557775	0.530016	0.279263	0.603473	0.579704	0.368912	0.678284	0.534508	0.25
3273	0.672822	0.465486	0.539101	0.547807	0.579704	0.438152	0.338662	0.560655	0.34
3274	0.366197	0.664407	0.191490	0.465860	0.579704	0.387157	0.343638	0.622659	0.61
3275	0.562477	0.535635	0.280484	0.560259	0.579704	0.255266	0.534114	0.632478	0.16

3276 rows × 9 columns

Handling Imbalance with SMOTE.

In [34]:
```
potability_count = data["Potability"].value_counts()
fig = sp.make_subplots(rows=1, cols=2, specs=[[{'type':'xy'}, {'type':'domain'}]])
bar_trace = go.Bar(x=potability_count.index, y=potability_count.values , marker=dict(colo
fig.add_trace(bar_trace, row=1, col=1)
pie_trace = go.Pie(labels=potability_count.index, values=potability_count.values)
fig.add_trace(pie_trace, row=1, col=2)
```

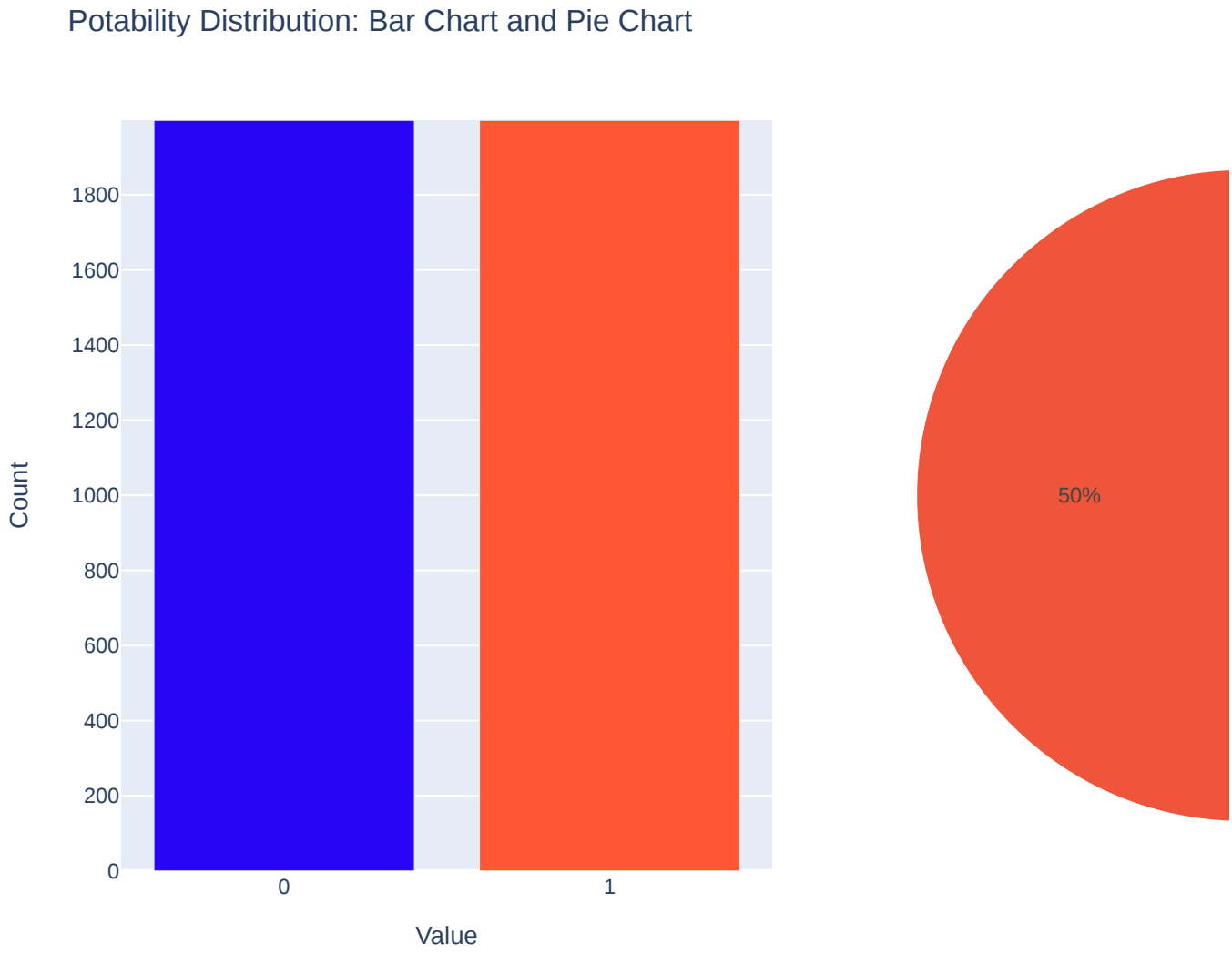
```
fig.update_layout(
    title_text="Potability Distribution: Bar Chart and Pie Chart",
    xaxis_title ="Value",
    yaxis_title="Count" ,
    xaxis=dict(tickmode='linear'),
    yaxis=dict(range=[0, max(potability_count.values) + 1]),
    width=1000,
    height=600
)

fig.show()
```



In [35]:
```
from imblearn.over_sampling import SMOTE
smote = SMOTE()
x , y = smote.fit_resample(x , y)
```

In [36]:
```
potability_count = y.value_counts()
fig = sp.make_subplots(rows=1, cols=2, specs=[[{'type':'xy'}, {'type':'domain'}]])
bar_trace = go.Bar(x=potability_count.index, y=potability_count.values , marker=dict(colo
fig.add_trace(bar_trace, row=1, col=1)
pie_trace = go.Pie(labels=potability_count.index, values=potability_count.values)
fig.add_trace(pie_trace, row=1, col=2)

fig.update_layout(
    title_text="Potability Distribution: Bar Chart and Pie Chart",
    xaxis_title ="Value",
    yaxis_title="Count" ,
```

```
    xaxis=dict(tickmode='linear'),
    yaxis=dict(range=[0, max(potability_count.values) + 1]),
    width=1000,
    height=600
)

fig.show()
```



- **Train & Test Split**

In [37]:
```
from sklearn.model_selection import train_test_split
x_train , x_test , y_train , y_test = train_test_split(x , y , test_size=0.2 , random_sta
```

Model Building

- # ML Classification Model Application
  - model1 = Support Vector Machine (SVM)
  - model2 = K Neighbors Classifier (KNN)

- model3 = Decision Tree Classifier
- model4 = Random Forest Classifier

In [38]:
```
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
model_1 = SVC()
model_2 = KNeighborsClassifier(n_neighbors=5)
model_3 = DecisionTreeClassifier(random_state=42)
model_4 = RandomForestClassifier(max_depth=10 , n_estimators=100, random_state=42)
```

1. SVC Model

In [39]:
```
svc = model_1.fit(x_train , y_train)
y_pred_train_svc = svc.predict(x_train)
y_pred_test_svc = svc.predict(x_test)
```

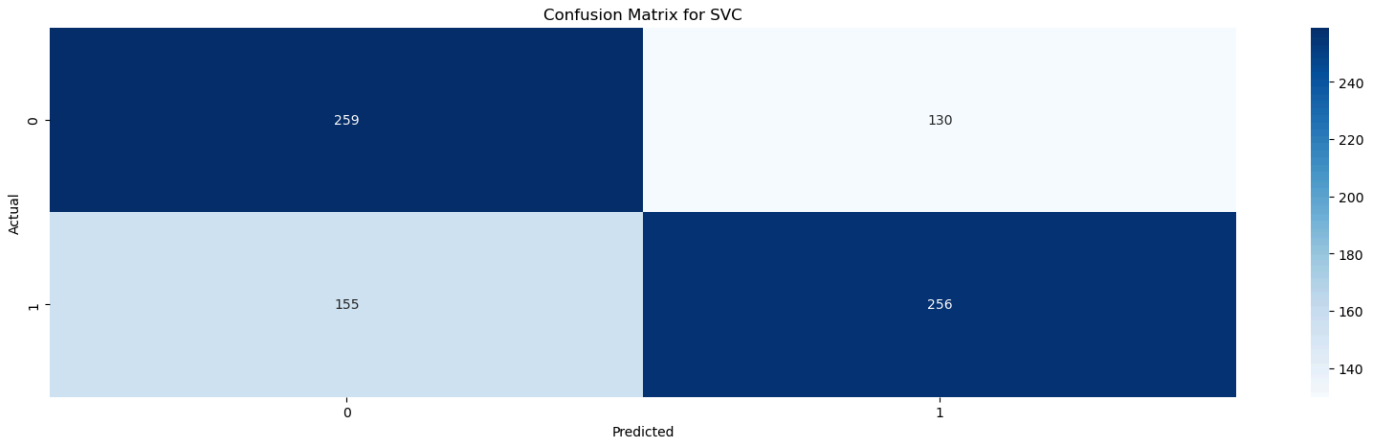
In [40]:
```
from sklearn.metrics import accuracy_score , classification_report , confusion_matrix
from sklearn import metrics
accuracy_train = accuracy_score(y_train , y_pred_train_svc)
accuracy_test = accuracy_score(y_test , y_pred_test_svc)
print(f"The Training Accuracy :{accuracy_train*100 :0.4} %")
print(f"The Testing Accuracy :{accuracy_test*100 :0.4} %")
```

```
The Training Accuracy :72.65 %
The Testing Accuracy :64.38 %
```

In [41]:
```
conf_matrix_svc = confusion_matrix(y_test,y_pred_test_svc)
conf_matrix_svc
```

Out[41]:
```
array([[259, 130],
       [155, 256]], dtype=int64)
```

In [42]:
```
plt.figure(figsize=(20,5))
sns.heatmap(conf_matrix_svc, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for SVC')
plt.show()
```

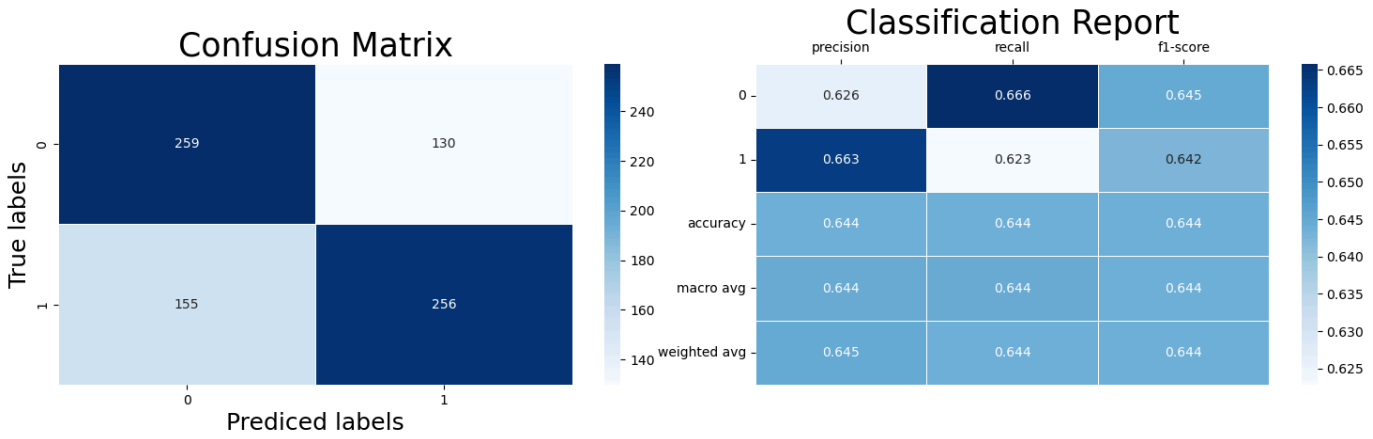


In [43]:
```
print(classification_report(y_test, y_pred_test_svc))
```

```
              precision    recall  f1-score   support

           0       0.63      0.67      0.65       389
           1       0.66      0.62      0.64       411

    accuracy                           0.64       800
   macro avg       0.64      0.64      0.64       800
weighted avg       0.64      0.64      0.64       800
```

In [44]:
```python
def plot_result(y_pred) :
    '''
    1) plot Confusion Matrix
    2) plot Classification Report
    '''
    fig, ax = plt.subplots(1, 2, figsize=(15, 4))
    fig.tight_layout()
    #AX left - Confusion Matrix
    cm = metrics.confusion_matrix(y_test, y_pred)
    ax[0]=sns.heatmap(cm, cmap='Blues', annot=True, fmt='', linewidths=0.5, ax=ax[0])
    ax[0].set_xlabel('Prediced labels', fontsize=18)
    ax[0].set_ylabel('True labels', fontsize=18)
    ax[0].set_title('Confusion Matrix', fontsize=25)
    ax[0].xaxis.set_ticklabels(['0', '1'])
    ax[0].yaxis.set_ticklabels(['0', '1'])
    #
    # AX Right - Classification Report
    cr = pd.DataFrame(metrics.classification_report(y_test, y_pred, digits=3, output_dict
    cr.drop(columns='support', inplace=True)
    ax[1] = sns.heatmap(cr, cmap='Blues', annot=True, fmt='0.3f', linewidths=0.5, ax=ax[1
    ax[1].xaxis.tick_top()
    ax[1].set_title('Classification Report', fontsize=25)
    plt.show()
```

In [45]:
```python
plot_result(y_pred_test_svc)
```



2. KNN Model

In [46]:
```python
KNN = model_2.fit(x_train , y_train)
y_pred_train_KNN = KNN.predict(x_train)
y_pred_test_KNN = KNN.predict(x_test)
```

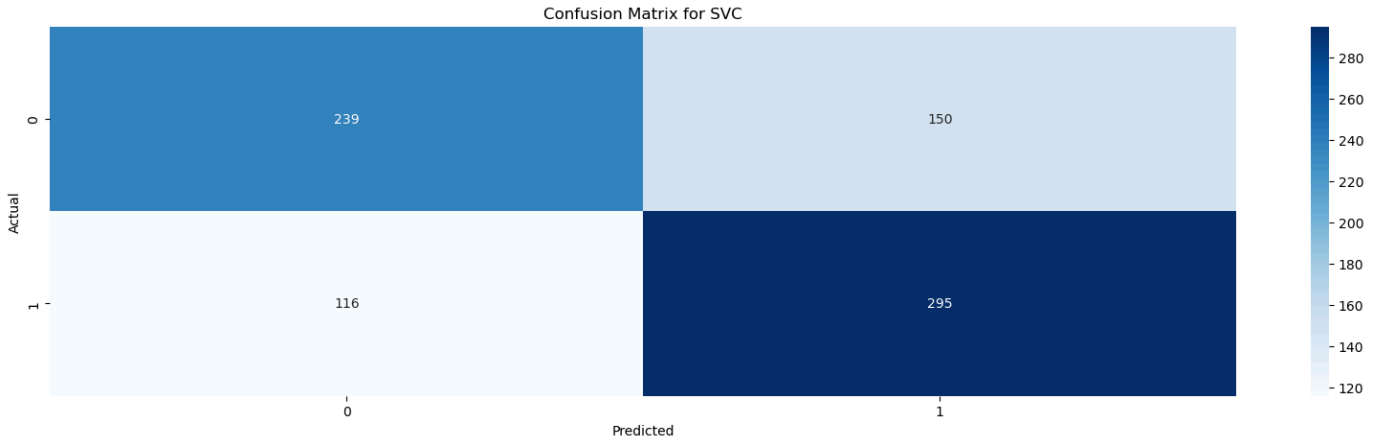
In [47]:
```python
from sklearn.metrics import accuracy_score , classification_report , confusion_matrix
from sklearn import metrics
accuracy_train = accuracy_score(y_train , y_pred_train_KNN)
accuracy_test = accuracy_score(y_test , y_pred_test_KNN)
print(f"The Training Accuracy :{accuracy_train*100 :0.4} %")
print(f"The Testing Accuracy :{accuracy_test*100 :0.4} %")
```

```
The Training Accuracy :78.75 %
The Testing Accuracy :66.75 %
```

In [48]:
```
conf_matrix_KNN = confusion_matrix(y_test,y_pred_test_KNN)
conf_matrix_KNN
```

Out[48]:
```
array([[239, 150],
       [116, 295]], dtype=int64)
```

In [49]:
```
plt.figure(figsize=(20,5))
sns.heatmap(conf_matrix_KNN, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for SVC')
plt.show()
```

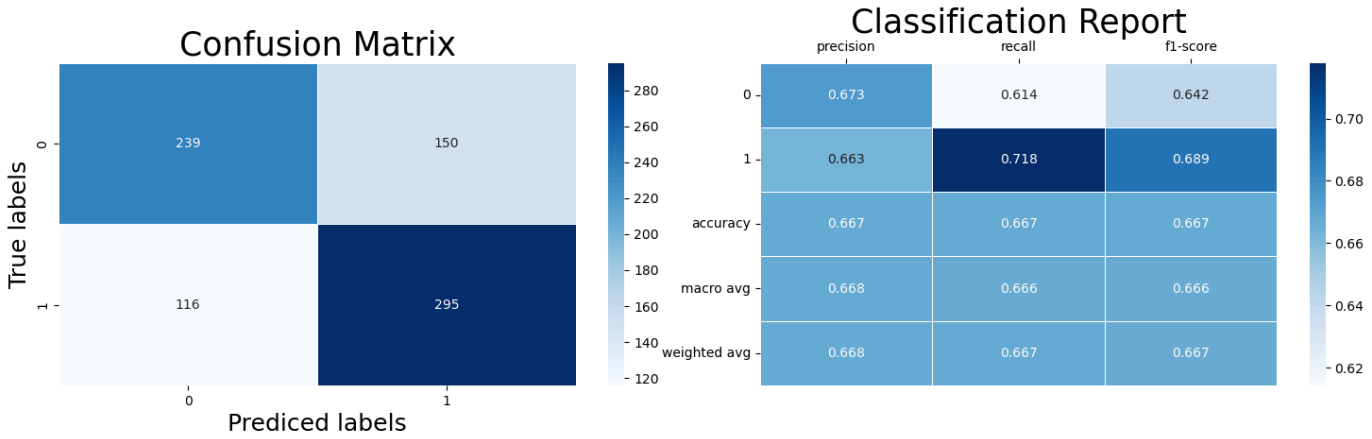


In [50]:
```
print(classification_report(y_test, y_pred_test_KNN))
```

```
              precision    recall  f1-score   support

           0       0.67      0.61      0.64       389
           1       0.66      0.72      0.69       411

    accuracy                           0.67       800
   macro avg       0.67      0.67      0.67       800
weighted avg       0.67      0.67      0.67       800
```

In [51]:
```
def plot_result(y_pred) :
    '''
    1) plot Confusion Matrix
    2) plot Classification Report
    '''
    fig, ax = plt.subplots(1, 2, figsize=(15, 4))
    fig.tight_layout()
    #AX left - Confusion Matrix
    cm = metrics.confusion_matrix(y_test, y_pred)
    ax[0]=sns.heatmap(cm, cmap='Blues', annot=True, fmt='', linewidths=0.5, ax=ax[0])
    ax[0].set_xlabel('Prediced labels', fontsize=18)
    ax[0].set_ylabel('True labels', fontsize=18)
    ax[0].set_title('Confusion Matrix', fontsize=25)
    ax[0].xaxis.set_ticklabels(['0', '1'])
    ax[0].yaxis.set_ticklabels(['0', '1'])
    #
    # AX Right - Classification Report
    cr = pd.DataFrame(metrics.classification_report(y_test, y_pred, digits=3, output_dict
    cr.drop(columns='support', inplace=True)
    ax[1] = sns.heatmap(cr, cmap='Blues', annot=True, fmt='0.3f', linewidths=0.5, ax=ax[1
    ax[1].xaxis.tick_top()
    ax[1].set_title('Classification Report', fontsize=25)
    plt.show()
```

In [52]:
```
plot_result(y_pred_test_KNN)
```



3. Decision Tree

In [53]:
```
decisiontree = model_3.fit(x_train , y_train)
y_pred_train_DT = decisiontree.predict(x_train)
y_pred_test_DT = decisiontree.predict(x_test)
```

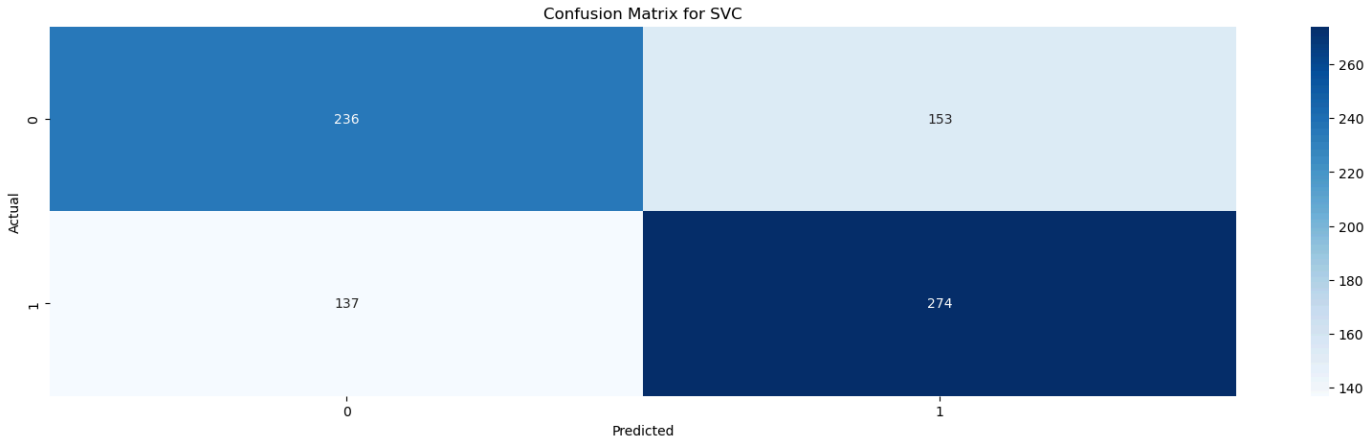
In [54]:
```
from sklearn.metrics import accuracy_score , classification_report , confusion_matrix
from sklearn import metrics
accuracy_train = accuracy_score(y_train , y_pred_train_DT)
accuracy_test = accuracy_score(y_test , y_pred_test_DT)
print(f"The Training Accuracy :{accuracy_train*100 :0.4} %")
print(f"The Testing Accuracy :{accuracy_test*100 :0.4} %")
```

```
The Training Accuracy :100.0 %
The Testing Accuracy :63.75 %
```

In [55]:
```
conf_matrix_DT = confusion_matrix(y_test,y_pred_test_DT)
conf_matrix_DT
```

Out[55]:
```
array([[236, 153],
       [137, 274]], dtype=int64)
```

In [56]:
```
plt.figure(figsize=(20,5))
sns.heatmap(conf_matrix_DT, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for SVC')
plt.show()
```

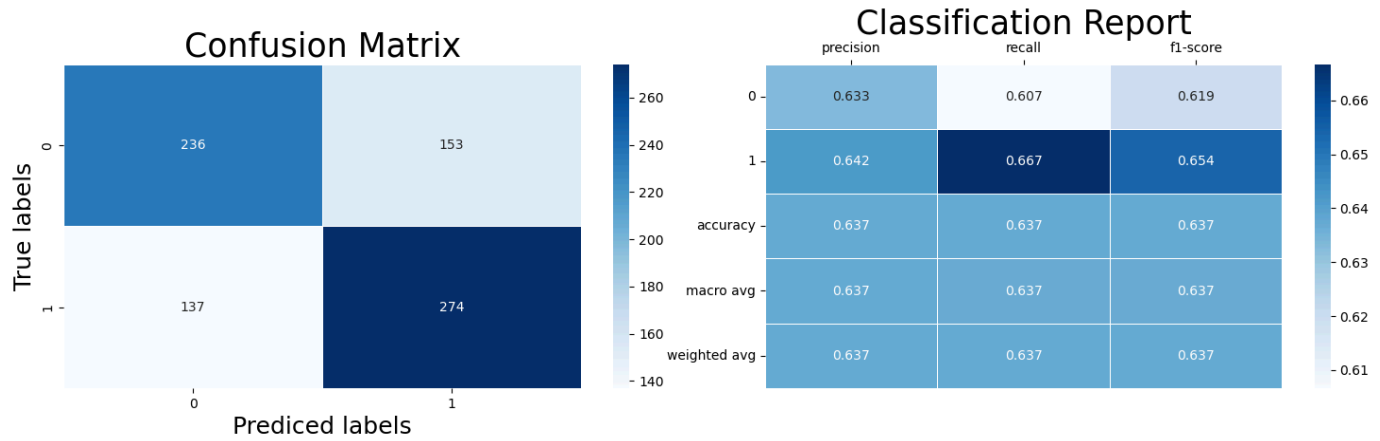


In [57]:
```
print(classification_report(y_test, y_pred_test_KNN))
```

```
              precision    recall  f1-score   support

           0       0.67      0.61      0.64       389
           1       0.66      0.72      0.69       411

    accuracy                           0.67       800
   macro avg       0.67      0.67      0.67       800
weighted avg       0.67      0.67      0.67       800
```

In [58]:
```python
def plot_result(y_pred) :
    '''
    1) plot Confusion Matrix
    2) plot Classification Report
    '''
    fig, ax = plt.subplots(1, 2, figsize=(15, 4))
    fig.tight_layout()
    #AX left - Confusion Matrix
    cm = metrics.confusion_matrix(y_test, y_pred)
    ax[0]=sns.heatmap(cm, cmap='Blues', annot=True, fmt='', linewidths=0.5, ax=ax[0])
    ax[0].set_xlabel('Prediced labels', fontsize=18)
    ax[0].set_ylabel('True labels', fontsize=18)
    ax[0].set_title('Confusion Matrix', fontsize=25)
    ax[0].xaxis.set_ticklabels(['0', '1'])
    ax[0].yaxis.set_ticklabels(['0', '1'])
    #
    # AX Right - Classification Report
    cr = pd.DataFrame(metrics.classification_report(y_test, y_pred, digits=3, output_dict
    cr.drop(columns='support', inplace=True)
    ax[1] = sns.heatmap(cr, cmap='Blues', annot=True, fmt='0.3f', linewidths=0.5, ax=ax[1
    ax[1].xaxis.tick_top()
    ax[1].set_title('Classification Report', fontsize=25)
    plt.show()
```

In [59]:
```python
plot_result(y_pred_test_DT)
```



4. Random Forest Classifier Model

In [60]:
```python
Random_forest = model_4.fit(x_train , y_train)
y_pred_train_RF = Random_forest.predict(x_train)
y_pred_test_RF = Random_forest.predict(x_test)
```

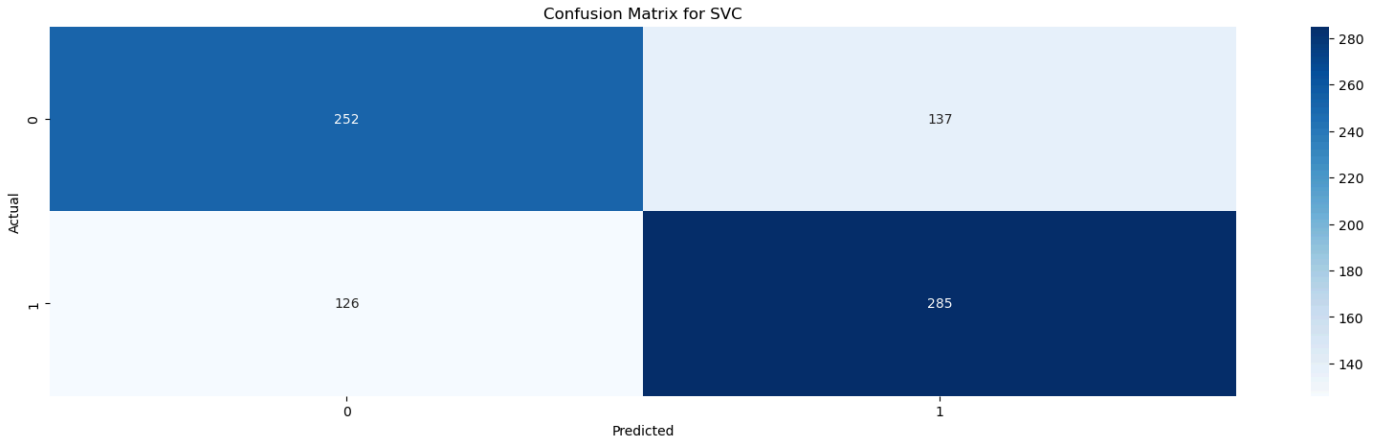
In [61]:
```python
from sklearn.metrics import accuracy_score , classification_report , confusion_matrix
from sklearn import metrics
accuracy_train = accuracy_score(y_train , y_pred_train_RF)
accuracy_test = accuracy_score(y_test , y_pred_test_RF)
print(f"The Training Accuracy :{accuracy_train*100 :0.4} %")
print(f"The Testing Accuracy :{accuracy_test*100 :0.4} %")
```

```
The Training Accuracy :92.83 %
The Testing Accuracy :67.12 %
```

In [62]:
```
conf_matrix_RF = confusion_matrix(y_test,y_pred_test_RF)
conf_matrix_RF
```

Out[62]:
```
array([[252, 137],
       [126, 285]], dtype=int64)
```

In [63]:
```
plt.figure(figsize=(20,5))
sns.heatmap(conf_matrix_RF, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix for SVC')
plt.show()
```

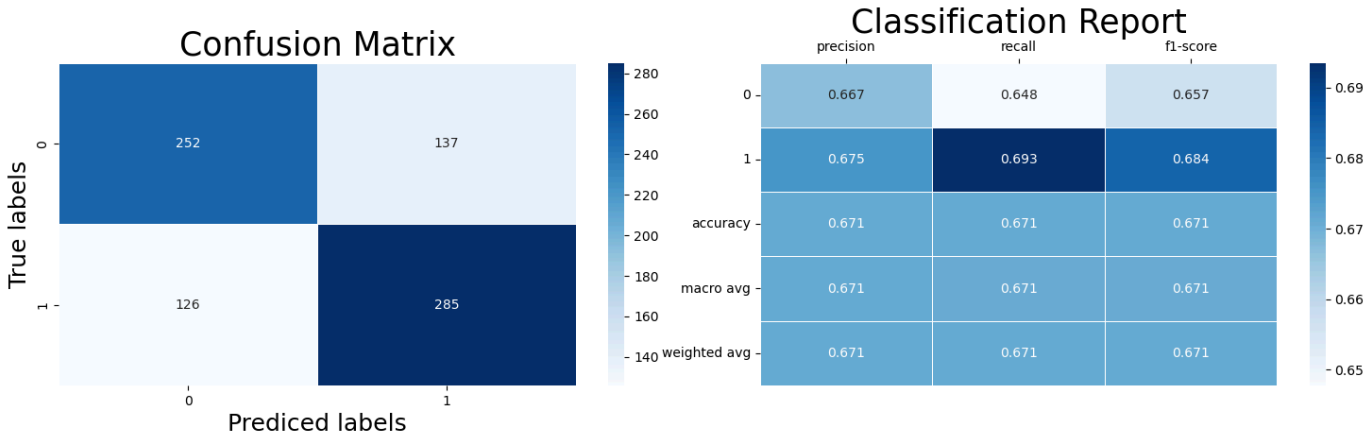


In [64]:
```
print(classification_report(y_test, y_pred_test_RF))
```

```
              precision    recall  f1-score   support

           0       0.67      0.65      0.66       389
           1       0.68      0.69      0.68       411

    accuracy                           0.67       800
   macro avg       0.67      0.67      0.67       800
weighted avg       0.67      0.67      0.67       800
```

In [65]:
```
def plot_result(y_pred) :
    '''
    1) plot Confusion Matrix
    2) plot Classification Report
    '''
    fig, ax = plt.subplots(1, 2, figsize=(15, 4))
    fig.tight_layout()
    #AX left - Confusion Matrix
    cm = metrics.confusion_matrix(y_test, y_pred)
    ax[0]=sns.heatmap(cm, cmap='Blues', annot=True, fmt='', linewidths=0.5, ax=ax[0])
    ax[0].set_xlabel('Prediced labels', fontsize=18)
    ax[0].set_ylabel('True labels', fontsize=18)
    ax[0].set_title('Confusion Matrix', fontsize=25)
    ax[0].xaxis.set_ticklabels(['0', '1'])
    ax[0].yaxis.set_ticklabels(['0', '1'])
    #
    # AX Right - Classification Report
    cr = pd.DataFrame(metrics.classification_report(y_test, y_pred, digits=3, output_dict
    cr.drop(columns='support', inplace=True)
    ax[1] = sns.heatmap(cr, cmap='Blues', annot=True, fmt='0.3f', linewidths=0.5, ax=ax[1
    ax[1].xaxis.tick_top()
    ax[1].set_title('Classification Report', fontsize=25)
    plt.show()
```

In [66]:
```
plot_result(y_pred_test_RF)
```



➡ **Best algorithm is RandomForestClassifier with score=0.688**