1. Import dependencies:

```
In [ ]: import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.utils import resample
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import svm
from sklearn import metrics
import pickle
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_3880\3193547673.py:2: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major release of pan
das (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and better inte
roperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54466

  import pandas as pd
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-
packages\keras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy i
s deprecated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

2. generate data:

```
In [ ]: # Définition des paramètres de l'eau (normes)
def generate_water_params(num_samples):
    pH = np.random.uniform(6.5, 8.5, num_samples)
    turbidity = np.random.uniform(0, 5, num_samples)  # Ajusté pour inclure plus de
    dissolved_oxygen = np.random.uniform(0, 10, num_samples)
    temperature = np.random.uniform(5, 25, num_samples)  # Ajusté pour inclure plus
    conductivity = np.random.uniform(0, 2000, num_samples)  # Ajusté pour inclure p

    return np.column_stack((pH, turbidity, dissolved_oxygen, temperature, conductiv

# Fonction pour étiqueter les données synthétiques en fonction des paramètres de qu
def classify_water_quality(params):
    labels = []
    for param in params:
        pH, turbidity, dissolved_oxygen, temperature, conductivity = param

        if 6.5 <= pH <= 8.5:
            ph_quality = 'normal'
        else:
            ph_quality = 'anormal'
```

```python
        if turbidity < 1:
            turbidity_quality = 'normal'
        else:
            turbidity_quality = 'anormal'

        if 8 <= dissolved_oxygen <= 10:
            do_quality = 'non polluée'
        elif 6 <= dissolved_oxygen < 8:
            do_quality = 'quasi polluée'
        elif 4 <= dissolved_oxygen < 6:
            do_quality = 'très polluée'
        else:
            do_quality = 'dangereux'

        if 6 <= temperature <= 20:
            temperature_quality = 'normal'
        else:
            temperature_quality = 'anormal'

        if 50 <= conductivity <= 1500:
            conductivity_quality = 'normal'
        else:
            conductivity_quality = 'anormal'

        # Combiner les résultats pour déterminer la classe finale
        if ph_quality == 'normal' and turbidity_quality == 'normal' and do_quality
            labels.append('eau non polluée')
        elif ph_quality == 'normal' and turbidity_quality == 'normal' and do_qualit
            labels.append('quasi polluée')
        elif ph_quality == 'normal' and turbidity_quality == 'normal' and do_qualit
            labels.append('très polluée')
        else:
            labels.append('dangereux')

    return labels

# Générer des données synthétiques
num_samples = 50000  # Générer plus de données pour permettre l'équilibrage
params = generate_water_params(num_samples)
labels = classify_water_quality(params)

# Créer un DataFrame
data = pd.DataFrame(params, columns=['pH', 'Turbidity', 'Dissolved Oxygen', 'Temper
data['Water Quality'] = labels

# Équilibrer les classes
eau_non_polluee = data[data['Water Quality'] == 'eau non polluée']
quasi_polluee = data[data['Water Quality'] == 'quasi polluée']
tres_polluee = data[data['Water Quality'] == 'très polluée']
dangereux = data[data['Water Quality'] == 'dangereux']

# Définir la taille cible pour chaque classe
target_size = min(len(eau_non_polluee), len(quasi_polluee), len(tres_polluee), len(

# Sous-échantillonner les classes majoritaires et sur-échantillonner les classes mi
```

```
eau_non_polluee_resampled = resample(eau_non_polluee, replace=True, n_samples=targe
quasi_polluee_resampled = resample(quasi_polluee, replace=True, n_samples=target_si
tres_polluee_resampled = resample(tres_polluee, replace=True, n_samples=target_size
dangereux_resampled = resample(dangereux, replace=False, n_samples=target_size, ran

# Combiner les échantillons équilibrés
balanced_data = pd.concat([eau_non_polluee_resampled, quasi_polluee_resampled, tres

# Compter chaque valeur de 'Water Quality' pour vérifier l'équilibrage
value_counts = balanced_data['Water Quality'].value_counts()
print(value_counts)

# Préparation des données pour l'entraînement
X = balanced_data[['pH', 'Turbidity', 'Dissolved Oxygen', 'Temperature', 'Conductiv
y = np.array([0 if label == 'eau non polluée' else 1 if label == 'quasi polluée' el

# Diviser les données en ensembles d'entraînement et de test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Modèle génératif simple (MLP)
model = Sequential([
    Dense(64, activation='relu', input_dim=5),
    Dense(64, activation='relu'),
    Dense(1, activation='linear')
])

model.compile(optimizer='adam', loss='mse')

# Entraîner le modèle sur les données d'entraînement
model.fit(X_train, y_train, epochs=50, batch_size=32)

# Prédire sur l'ensemble de test
y_pred = model.predict(X_test)
y_pred_labels = ['eau non polluée' if label < 0.5 else 'quasi polluée' if label < 1
y_test_labels = ['eau non polluée' if label == 0 else 'quasi polluée' if label == 1

# Calculer l'accuracy
accuracy = accuracy_score(y_test_labels, y_pred_labels)
print(f'Accuracy: {accuracy}')

# Générer de nouvelles données synthétiques
new_samples = 500000
new_params = generate_water_params(new_samples)
new_labels = model.predict(new_params)

# Convertir les prédictions en catégories
new_labels = ['eau non polluée' if label < 0.5 else 'quasi polluée' if label < 1.5

# Créer un nouveau DataFrame avec les données générées
new_data = pd.DataFrame(new_params, columns=['pH', 'Turbidity', 'Dissolved Oxygen',
new_data['Water Quality'] = new_labels

# Sauvegarder les données générées dans un fichier CSV
new_data.to_csv('generated_water_quality_data.csv', index=False)
print("Données générées et sauvegardées dans 'generated_water_quality_data.csv'")
```

```
Water Quality
eau non polluée    997
quasi polluée      997
très polluée       997
dangereux          997
Name: count, dtype: int64
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\backend.py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer instead.

Epoch 1/50
WARNING:tensorflow:From c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTensorValue instead.

100/100 [==============================] - 1s 2ms/step - loss: 76.3738
Epoch 2/50
100/100 [==============================] - 0s 2ms/step - loss: 0.6584
Epoch 3/50
100/100 [==============================] - 0s 2ms/step - loss: 0.6414
Epoch 4/50
100/100 [==============================] - 0s 2ms/step - loss: 0.5953
Epoch 5/50
100/100 [==============================] - 0s 2ms/step - loss: 0.6135
Epoch 6/50
100/100 [==============================] - 0s 2ms/step - loss: 0.7250
Epoch 7/50
100/100 [==============================] - 0s 2ms/step - loss: 0.7441
Epoch 8/50
100/100 [==============================] - 0s 2ms/step - loss: 0.6828
Epoch 9/50
100/100 [==============================] - 0s 2ms/step - loss: 0.7609
Epoch 10/50
100/100 [==============================] - 0s 2ms/step - loss: 0.8059
Epoch 11/50
100/100 [==============================] - 0s 2ms/step - loss: 1.2415
Epoch 12/50
100/100 [==============================] - 0s 2ms/step - loss: 1.0013
Epoch 13/50
100/100 [==============================] - 0s 2ms/step - loss: 0.8252
Epoch 14/50
100/100 [==============================] - 0s 2ms/step - loss: 1.8300
Epoch 15/50
100/100 [==============================] - 0s 2ms/step - loss: 1.2352
Epoch 16/50
100/100 [==============================] - 0s 2ms/step - loss: 6.4045
Epoch 17/50
100/100 [==============================] - 0s 2ms/step - loss: 2.5553
Epoch 18/50
100/100 [==============================] - 0s 2ms/step - loss: 1.5973
Epoch 19/50
100/100 [==============================] - 0s 2ms/step - loss: 0.7969
```

```
Epoch 20/50
100/100 [==============================] - 0s 2ms/step - loss: 1.1556
Epoch 21/50
100/100 [==============================] - 0s 2ms/step - loss: 1.0639
Epoch 22/50
100/100 [==============================] - 0s 2ms/step - loss: 2.4729
Epoch 23/50
100/100 [==============================] - 0s 2ms/step - loss: 2.0492
Epoch 24/50
100/100 [==============================] - 0s 2ms/step - loss: 3.5671
Epoch 25/50
100/100 [==============================] - 0s 2ms/step - loss: 1.4848
Epoch 26/50
100/100 [==============================] - 0s 2ms/step - loss: 0.7260
Epoch 27/50
100/100 [==============================] - 0s 2ms/step - loss: 1.6782
Epoch 28/50
100/100 [==============================] - 0s 2ms/step - loss: 1.1486
Epoch 29/50
100/100 [==============================] - 0s 2ms/step - loss: 1.1997
Epoch 30/50
100/100 [==============================] - 0s 2ms/step - loss: 1.5337
Epoch 31/50
100/100 [==============================] - 0s 2ms/step - loss: 6.0538
Epoch 32/50
100/100 [==============================] - 0s 2ms/step - loss: 0.8748
Epoch 33/50
100/100 [==============================] - 0s 2ms/step - loss: 0.9868
Epoch 34/50
100/100 [==============================] - 0s 2ms/step - loss: 1.7605
Epoch 35/50
100/100 [==============================] - 0s 2ms/step - loss: 1.6684
Epoch 36/50
100/100 [==============================] - 0s 2ms/step - loss: 3.2311
Epoch 37/50
100/100 [==============================] - 0s 2ms/step - loss: 0.9649
Epoch 38/50
100/100 [==============================] - 0s 2ms/step - loss: 1.1601
Epoch 39/50
100/100 [==============================] - 0s 2ms/step - loss: 1.5953
Epoch 40/50
100/100 [==============================] - 0s 2ms/step - loss: 14.1101
Epoch 41/50
100/100 [==============================] - 0s 2ms/step - loss: 0.8619
Epoch 42/50
100/100 [==============================] - 0s 2ms/step - loss: 0.8437
Epoch 43/50
100/100 [==============================] - 0s 2ms/step - loss: 0.7634
Epoch 44/50
100/100 [==============================] - 0s 2ms/step - loss: 1.0484
Epoch 45/50
100/100 [==============================] - 0s 2ms/step - loss: 0.8648
Epoch 46/50
100/100 [==============================] - 0s 2ms/step - loss: 0.9096
Epoch 47/50
100/100 [==============================] - 0s 2ms/step - loss: 1.1054
```

```
Epoch 48/50
100/100 [==============================] - 0s 2ms/step - loss: 0.7312
Epoch 49/50
100/100 [==============================] - 0s 2ms/step - loss: 1.2752
Epoch 50/50
100/100 [==============================] - 0s 2ms/step - loss: 3.3759
25/25 [==============================] - 0s 1ms/step
Accuracy: 0.30451127819548873
15625/15625 [==============================] - 24s 2ms/step
Données générées et sauvegardées dans 'generated_water_quality_data.csv'
```

3. Data Collection:

A- Loading data:

In [ ]:
```
data = pd.read_csv("generated_water_quality_data.csv")
```

B- Head of the data:

In [ ]:
```
data.head()
```

Out[ ]:

	pH	Turbidity	Dissolved Oxygen	Temperature	Conductivity	Water Quality
0	6.804416	3.360733	6.568877	21.820872	487.616772	quasi polluée
1	7.605819	2.810029	1.497889	16.327887	721.284413	quasi polluée
2	7.168846	1.887626	8.828366	12.455322	1449.840041	eau non polluée
3	7.060933	3.445414	3.900543	10.981174	951.528088	eau non polluée
4	7.833177	0.190726	0.032128	24.058834	1475.002280	eau non polluée

C- Number of row and columns:

In [ ]:
```
len(data)
```

Out[ ]:
```
500000
```

In [ ]:
```
value_counts = data['Water Quality'].value_counts()
value_counts
```

Out[ ]:
```
Water Quality
eau non polluée    339271
quasi polluée       73174
très polluée        56968
dangereux           30587
Name: count, dtype: int64
```

In [ ]:
```
# Sample 30,000 instances of each category
data = data.groupby('Water Quality').apply(lambda x: x.sample(n=30000, random_state

# Check the new counts
```

```
value_counts = data['Water Quality'].value_counts()
print(value_counts)
```

```
Water Quality
dangereux          30000
eau non polluée    30000
quasi polluée      30000
très polluée       30000
Name: count, dtype: int64
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_3880\511566930.py:2: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.
  data = data.groupby('Water Quality').apply(lambda x: x.sample(n=30000, random_state=42)).reset_index(drop=True)
```

4. Statisctical measures :

A- General statistics:

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

Out[ ]:

	pH	Turbidity	Dissolved Oxygen	Temperature	Conductivity
count	120000.000000	120000.000000	120000.000000	120000.000000	120000.000000
mean	7.532142	2.874471	4.111559	15.396202	588.342348
std	0.578144	1.412601	2.804065	5.760054	518.232826
min	6.500003	0.000022	0.000021	5.000047	0.003454
25%	7.035231	1.741017	1.667992	10.487950	185.035400
50%	7.547669	3.048297	3.704256	15.574509	424.220654
75%	8.037879	4.110419	6.304838	20.403409	847.129060
max	8.499990	4.999983	9.999983	24.999165	1999.967908

B- General statistics:

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

Out[ ]:
```
pH                  0
Turbidity           0
Dissolved Oxygen    0
Temperature         0
Conductivity        0
Water Quality       0
dtype: int64
```

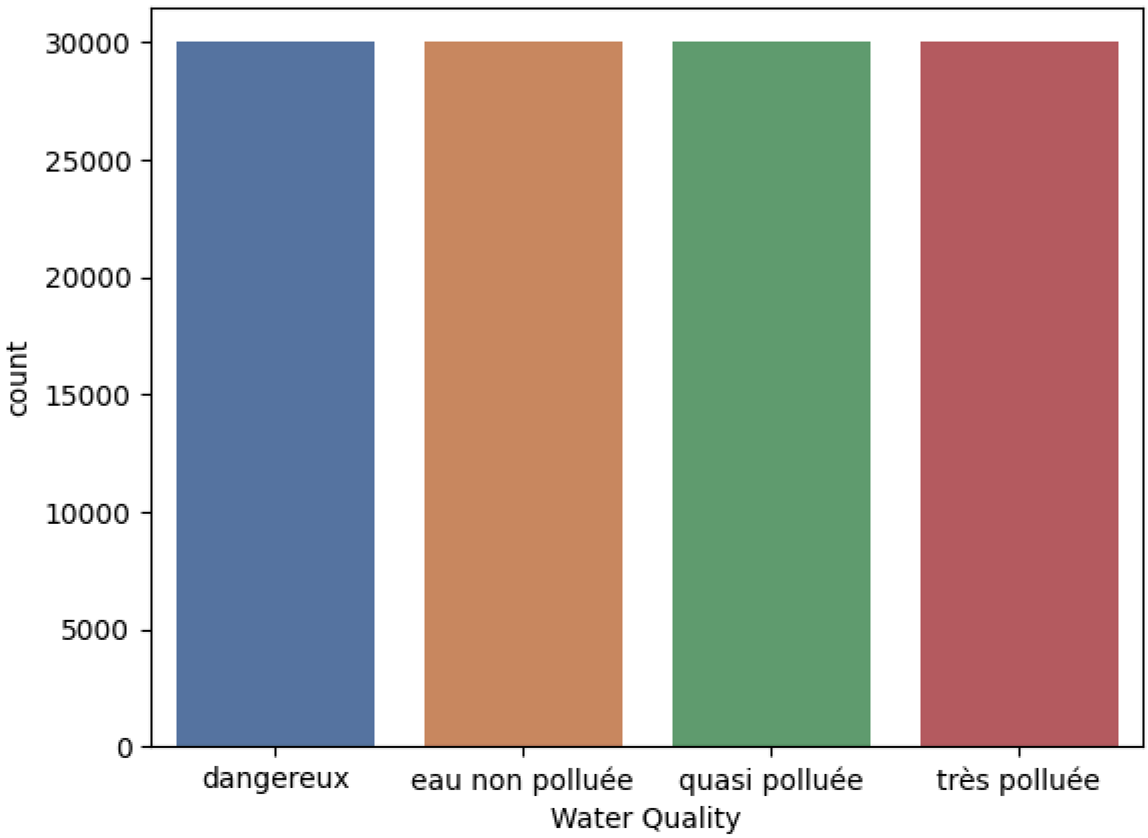
3. Visualisation:

A- The distrubution of the Water Quality columns:

In [ ]:
```
sns.countplot(data= data , x = 'Water Quality' , palette= 'deep')
```

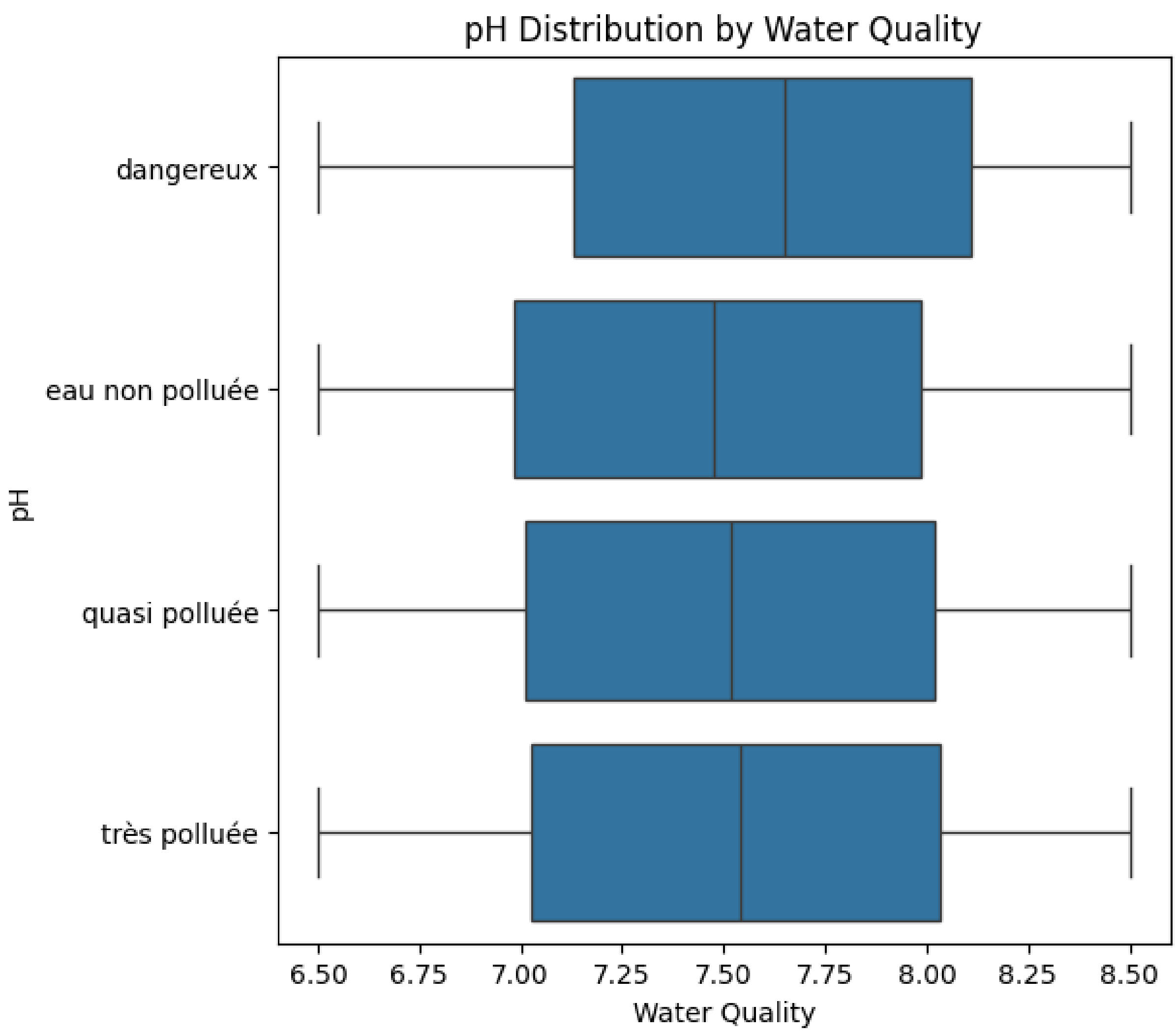
```
C:\Users\HP\AppData\Local\Temp\ipykernel_3284\3381706857.py:1: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.1
4.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  sns.countplot(data= data , x = 'Water Quality' , palette= 'deep')
```

Out[ ]: <Axes: xlabel='Water Quality', ylabel='count'>



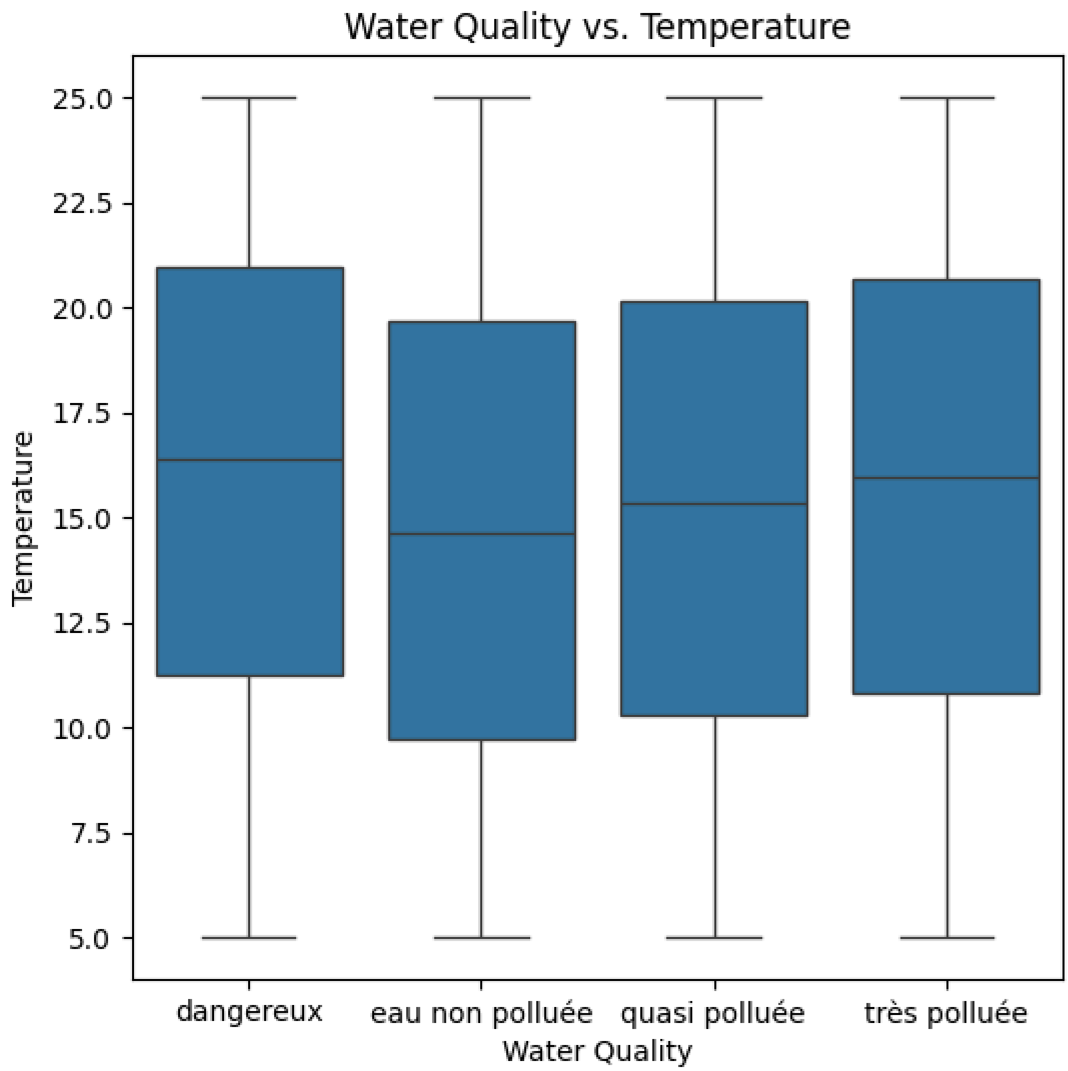
B- Ph Distribution by Water Quality:

In [ ]:
```
plt.figure(figsize=(6, 6))
sns.boxplot(x='pH', y='Water Quality', data=data)
plt.title('pH Distribution by Water Quality')
plt.xlabel('Water Quality')
plt.ylabel('pH')
plt.show()
```



C- Température Distribution by Water Quality:

```
In [ ]: plt.figure(figsize=(6, 6))
        sns.boxplot(x='Water Quality', y='Temperature', data=data)
        plt.title('Water Quality vs. Temperature')
        plt.xlabel('Water Quality')
        plt.ylabel('Temperature')
        plt.show()
```



4. Label Encoding:

A- Encoding the categorical data:

In [ ]:
```
data.replace({"Water Quality": {'eau non polluée':1,'quasi polluée':2,'très polluée
data.head()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_3880\1597305834.py:1: FutureWarning: Downca
sting behavior in `replace` is deprecated and will be removed in a future version. T
o retain the old behavior, explicitly call `result.infer_objects(copy=False)`. To op
t-in to the future behavior, set `pd.set_option('future.no_silent_downcasting', Tru
e)`
  data.replace({"Water Quality": {'eau non polluée':1,'quasi polluée':2,'très pollué
e':3,'dangereux':4 }}, inplace=True)
```

Out[ ]:

	pH	Turbidity	Dissolved Oxygen	Temperature	Conductivity	Water Quality
0	7.683379	3.486836	2.759840	11.275598	185.191065	4
1	6.588063	4.992089	3.724989	20.273811	295.294325	4
2	6.956671	3.961985	2.869450	15.543482	67.700295	4
3	7.699439	1.645307	0.651428	8.180875	69.634085	4
4	7.730778	4.422486	6.254852	6.182293	2.664449	4

In [ ]:

```
# Check the new counts
value_counts = data['Water Quality'].value_counts()
print(value_counts)
```

```
Water Quality
4    30000
1    30000
2    30000
3    30000
Name: count, dtype: int64
```

5. Train test split:

A- Separating data & lables:

In [ ]:

```
X = data.drop(columns= ['Water Quality'] , axis= 1)
Y = data['Water Quality']
```

In [ ]:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

B- Test Split

In [ ]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_s
```

In [ ]:

```
print(X.shape,X_train.shape, X_test.shape)
```

```
(120000, 5) (96000, 5) (24000, 5)
```

6. Training the model (SVM):

A- Creating the model:

In [ ]:

```
model = svm.SVC(kernel= 'linear')
```

B- Training the model:

In [ ]:

```
model.fit(X_train , Y_train)
```

Out[ ]:

```
SVC
SVC(kernel='linear')
```

7. Model Evaluation :

A- Accuracy score of training data:

In [ ]:

```python
X_train_prediction = model.predict(X_train)
data_accuracy = accuracy_score(X_train_prediction, Y_train)
print('Accuracy on training data : ', data_accuracy)
```

```
Accuracy on training data :  0.94484375
```

B- Accuracy score of testing data:

In [ ]:

```python
X_test_prediction = model.predict(X_test)
data_accuracy = accuracy_score(X_test_prediction, Y_test)
print('Accuracy on testing data : ', data_accuracy)
```

```
Accuracy on testing data :  0.9472916666666666
```

8. Example:

In [ ]:

```python
def prediction_water_quality(pH, turbidity, dissolved_oxygen, temperature, conducti
    input_data = (pH, turbidity, dissolved_oxygen, temperature, conductivity)
    # Convertir les données en tableau numpy
    input_dataNumpy = np.asarray(input_data).reshape(1, -1)
    # Normaliser les données
    input_dataNumpy = scaler.transform(input_dataNumpy)
    # Faire la prédiction
    prediction = model.predict(input_dataNumpy)
    test = prediction[0]

    if test == 1:
        result = 'eau non polluée'
    elif test == 2:
        result = 'quasi polluée'
    elif test == 3:
        result = 'très polluée'
    elif test == 4:
        result = 'dangereux'
    else:
        result = 'valeur de prédiction inattendue'

    return result

# Exemple d'utilisation
print("Welcome to our model")
test = prediction_water_quality(7.7872151367013185,0.28390376108336224,0.0245983640
print(test)
```

```
Welcome to our model
très polluée
```

```
c:\Users\HP\AppData\Local\Programs\Python\Python39\lib\site-packages\sklearn\base.p
y:493: UserWarning: X does not have valid feature names, but StandardScaler was fitt
ed with feature names
  warnings.warn(
```

9. Loading the model:

In [ ]:

```python
import joblib
joblib.dump(model, 'model.pkl')
joblib.dump(scaler, 'scaler.pkl')
```

Out[ ]:

```
['scaler.pkl']
```