Water Quality Assessment Using Fuzzy Logic

In [2]:
```
# Importing necessary libraries

import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
import seaborn as sns

import skfuzzy as fuzz
from skfuzzy import control as ctrl

from IPython.display import Image, display

import warnings
warnings.filterwarnings('ignore')
```

About the dataset

From: Central Ground Water Board (CGWB), Ministry of Jal Shakti, Department of Water Resources, River Development and Ganga Rejuvenation.

Description: Several Ground Water Quality parameters of locations across India in the year 2022.

Source: https://www.cgwb.gov.in/old_website/WQ/Water%20Quality/Water%20quality%20data%202022.pdf

In [4]:
```
# Reading the dataset
df_org=pd.read_excel(r"D:\Data Science\Projects\3. Fuzzy Logic for Water Quality Assessment Project\Water quality data 2022.xlsx")
df_org.head()
```

Out[4]:

	Sl_No	GEMS_ID	W	STATE	DISTRICT	BLOCK	LOCATION	LONGITUDE	LATITUDE	Year	pH	...	PO4	TH	Ca
0	1	W193530074180001		Maharashtra	Ahmednagar	SANGAMNER	Kokangaon	74.3	19.591667	2022	7.90	...	NaN	465.0	80.2
1	2	W192600074540001		Maharashtra	Ahmednagar	NEVASA	Vadala Bhairoba	74.9	19.433333	2022	7.55	...	NaN	510.0	116.2
2	3	W192400075180001		Maharashtra	Ahmednagar	SHEVGAON	Ghotan	75.3	19.4	2022	7.36	...	NaN	1900.0	348.7
3	4	W191640074134001		Maharashtra	Ahmednagar	SANGAMNER	Bote	74.227778	19.277778	2022	7.97	...	NaN	425.0	76.2
4	5	W193130073553001		Maharashtra	Ahmednagar	AKOLA	Rajur	73.925	19.525	2022	7.85	...	NaN	290.0	50.1

5 rows × 26 columns

In [5]:
```
df_org.shape
```

Out[5]:
```
(13600, 26)
```

In [6]:
```
df_org.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13600 entries, 0 to 13599
Data columns (total 26 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Sl_No         13600 non-null  int64
 1   GEMS_ID    W  11366 non-null  object
 2   STATE         13600 non-null  object
 3   DISTRICT      13600 non-null  object
 4   BLOCK         13520 non-null  object
 5   LOCATION      13600 non-null  object
 6   LONGITUDE     13600 non-null  object
 7   LATITUDE      13600 non-null  object
 8   Year          13600 non-null  int64
 9   pH            13600 non-null  float64
 10  EC            13600 non-null  float64
 11  CO3           13600 non-null  int64
 12  HCO3          13600 non-null  float64
 13  Cl            13600 non-null  float64
 14  SO4           13600 non-null  float64
 15  NO3           13600 non-null  float64
 16  PO4           12242 non-null  float64
 17  TH            13600 non-null  float64
 18  Ca            13600 non-null  float64
 19  Mg            13600 non-null  float64
 20  Na            13600 non-null  float64
 21  K             13600 non-null  float64
 22  F             13600 non-null  float64
 23  SiO2          0 non-null      float64
 24  TDS           0 non-null      float64
 25  U_ppb_        13600 non-null  int64
dtypes: float64(15), int64(4), object(7)
memory usage: 2.7+ MB
```

In [7]:
```
# Converting State and District names to all caps to avoid any discripancies
df_org["STATE"] = df_org["STATE"].str.upper()
df_org["DISTRICT"] = df_org["DISTRICT"].str.upper()
df_org.sample(5)
```

Out[7]:

	Sl_No	GEMS_ID W	STATE	DISTRICT	BLOCK	LOCATION	LONGITUDE	LATITUDE	Year	pH	...	PO4	
6824	6825	W11163177482901	TAMIL NADU	NAMAKKAL	THIRUCHENGODU	Patlur	77.8081	11.2753	2022	7.80	...	0.0	310
437	438	W191846077135901	MAHARASHTRA	HINGOLI	BASMATH	Palasgaon	77.233056	19.312778	2022	7.73	...	NaN	280
1200	1201	W191649072471101	MAHARASHTRA	THANE	VASAI	Uttan	72.786389	19.280278	2022	7.80	...	NaN	320
7598	7599	W212105083500201	ODISHA	BARGARH	ATTABIRA	Larambha	83.833889	21.351389	2022	7.87	...	0.0	194
12016	12017	NaN	PUNJAB	BATHINDA	Rampura phul	Dhapali	75.242	30.363	2022	8.25	...	0.0	52

5 rows × 26 columns

Handling Missing/Duplicate Values

In [9]:
```
# Checking missing values
df_org.isna().sum() # we won't be using columns with large no. of missing values
```

Out[9]:
```
Sl_No               0
GEMS_ID    W     2234
STATE               0
DISTRICT            0
BLOCK              80
LOCATION            0
LONGITUDE           0
LATITUDE            0
Year                0
pH                  0
EC                  0
CO3                 0
HCO3                0
Cl                  0
SO4                 0
NO3                 0
PO4              1358
TH                  0
Ca                  0
Mg                  0
Na                  0
K                   0
F                   0
SiO2            13600
TDS             13600
U_ppb_              0
dtype: int64
```

In [10]:
```
# Checking if any duplicated rows
df_org.duplicated().any()
```

Out[10]:
```
False
```

In [11]:
```
df_org.duplicated().sum() # no duplicate rows present
```

Out[11]:
```
0
```

Feature Extraction

In [13]:
```
df_org.columns
```

Out[13]:
```
Index(['Sl_No', 'GEMS_ID    W', 'STATE', 'DISTRICT', 'BLOCK', 'LOCATION',
       'LONGITUDE', 'LATITUDE', 'Year', 'pH', 'EC', 'CO3', 'HCO3', 'Cl', 'SO4',
       'NO3', 'PO4', 'TH', 'Ca', 'Mg', 'Na', 'K', 'F', 'SiO2', 'TDS',
       'U_ppb_'],
      dtype='object')
```

What is Water Quality Index (WQI):

WQI indicates the quality of water in terms of index number which represents overall quality of water for any intended use. It is defined as a rating reflecting the composite influence of different water quality parameters were taken into consideration for the calculation of water Quality index (WQI) .The indices are among the most effective ways to communicate the information on water quality trends to the general public or to the policy makers and in water quality management. In formulation of water quality index the relative importance of various parameters depends on intended use of water. Mostly it is done from the point of view of its suitability for human consumption.

In [15]:
```
# Using the following 4 columns for WQI calculations
df = df_org[['pH', 'TH', 'Cl', 'F']]  # 1. pH, 2. Total Hardness, 3. Chlorine Content, 4. Fluorine Content
```

In [16]:
```
df.head()
```

Out[16]:

	pH	TH	Cl	F
0	7.90	465.0	120.5	0.53
1	7.55	510.0	191.4	0.18
2	7.36	1900.0	1201.8	0.11
3	7.97	425.0	102.8	0.31
4	7.85	290.0	24.8	0.20

In [17]:
```
# Rounding-off the values to 1 decimal place for easier calculations
df = df.round(1)
df.tail()
```

Out[17]:

	pH	TH	Cl	F
13595	7.7	500.0	1077.7	2.0
13596	7.7	2900.0	5672.0	0.9
13597	8.1	380.0	567.2	3.8
13598	7.6	640.0	581.4	0.8
13599	8.2	380.0	1843.4	1.2

In [18]:
```
df.shape
```

Out[18]:
```
(13600, 4)
```

In [19]:
```
df.dtypes
```

Out[19]:
```
pH      float64
TH      float64
Cl      float64
F       float64
dtype: object
```

In [20]:
```
df.describe() # Checking max values of the parameters
```

Out[20]:

	pH	TH	Cl	F
count	13600.000000	13600.00000	13600.000000	13600.000000
mean	7.720081	323.54064	170.535596	0.647581
std	0.449209	293.74896	365.517279	2.179632
min	2.800000	3.00000	0.000000	0.000000
25%	7.500000	170.00000	28.400000	0.200000
50%	7.700000	262.15000	71.000000	0.400000
75%	8.000000	390.00000	170.200000	0.800000
max	10.900000	8606.90000	9358.800000	168.000000

Step Wise Process of Implementing Fuzzy Logic

BIS standards for drinking water

Parameter : Acceptable Range

1. pH : 6.5 - 8.5
2. Hardness : 200 - 600
3. Cl : 250 - 1000
4. F : 1 - 1.5

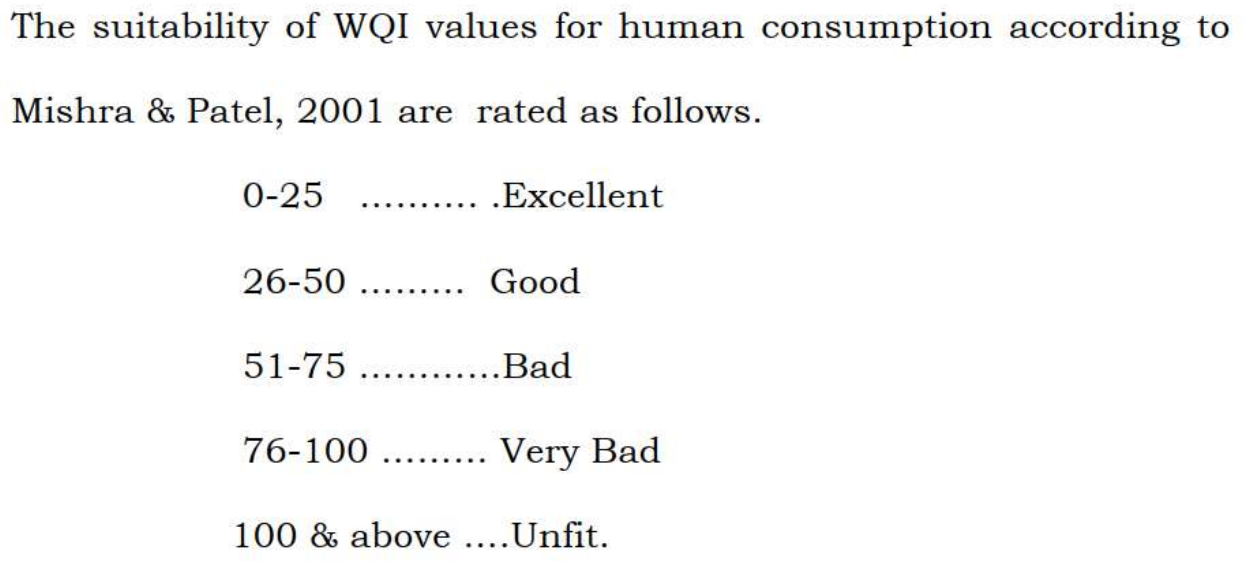
Source: https://cpcb.nic.in/wqm/BIS_Drinking_Water_Specification.pdf

In [23]:
```
# 1. Define fuzzy input variables
ph = ctrl.Antecedent(np.arange(0, 14, 0.1), 'pH')
th = ctrl.Antecedent(np.arange(0, 8700, 0.1), 'TH')
cl = ctrl.Antecedent(np.arange(0, 9500, 0.1), 'Cl')
f = ctrl.Antecedent(np.arange(0, 170, 0.1), 'F')

# 2. Define fuzzy output variable
wqi = ctrl.Consequent(np.arange(0, 201, 0.1), 'WQI')

# 3. Membership functions for inputs
ph['permissible'] = fuzz.trapmf(ph.universe, [6, 6.5, 8.5, 9])
ph['unacceptable'] = np.fmax(
    fuzz.trimf(ph.universe, [0, 5, 6.5]),
    fuzz.trimf(ph.universe, [8.5, 10, 14])
)

th['permissible'] = fuzz.trapmf(th.universe, [0, 200, 600, 650])
th['unacceptable'] = fuzz.trimf(th.universe, [600, 800, 8700])

cl['permissible'] = fuzz.trapmf(cl.universe, [0, 250, 1000, 1100])
cl['unacceptable'] = fuzz.trimf(cl.universe, [1000, 2000, 9500])
```

```
f['permissible'] = fuzz.trapmf(f.universe, [0.5, 1, 1.5, 2])
f['unacceptable'] = np.fmax(
    fuzz.trimf(f.universe, [0, 0.5, 1]),
    fuzz.trimf(f.universe, [1.5, 5, 170])
)
```

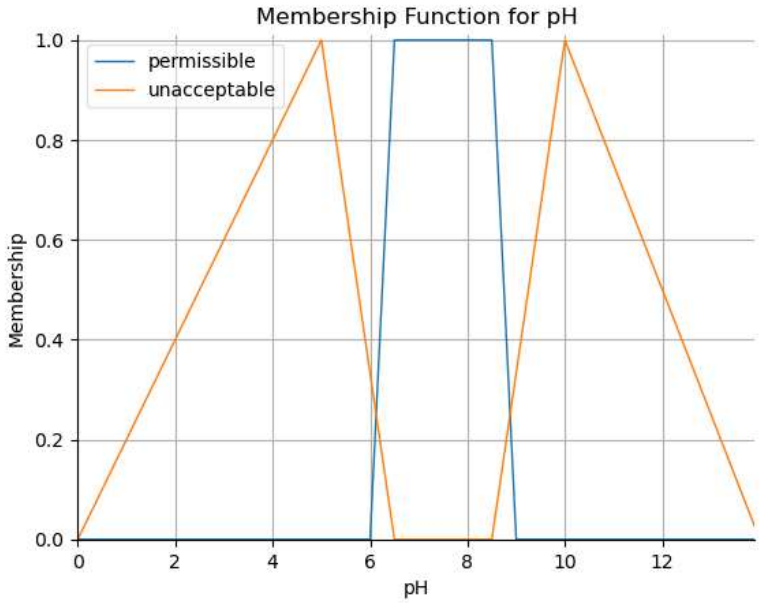
Water Quality Categories Classification Based on WQI

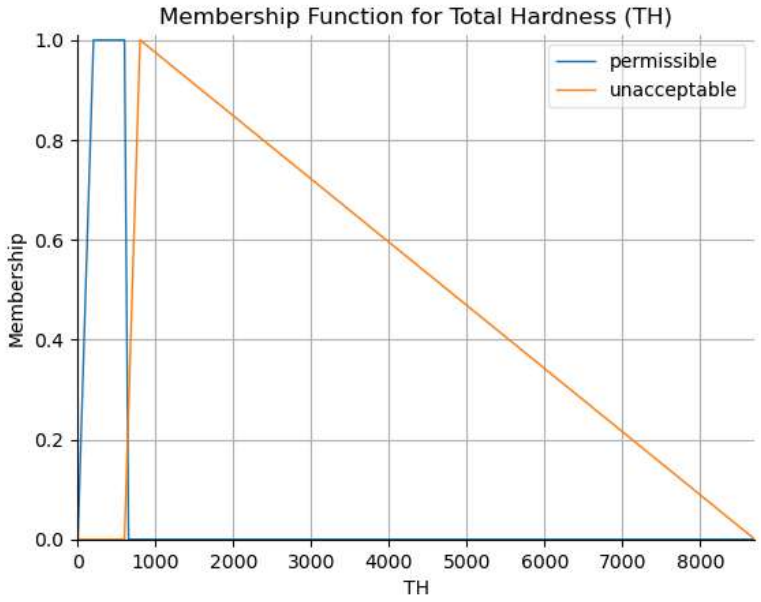
In [25]:
```
# Displaying the image for the WQI categories
display(Image(filename=r"D:\Data Science\Projects\3. Fuzzy Logic for Water Quality Assessment Project\WQI_ranges.png"))
```

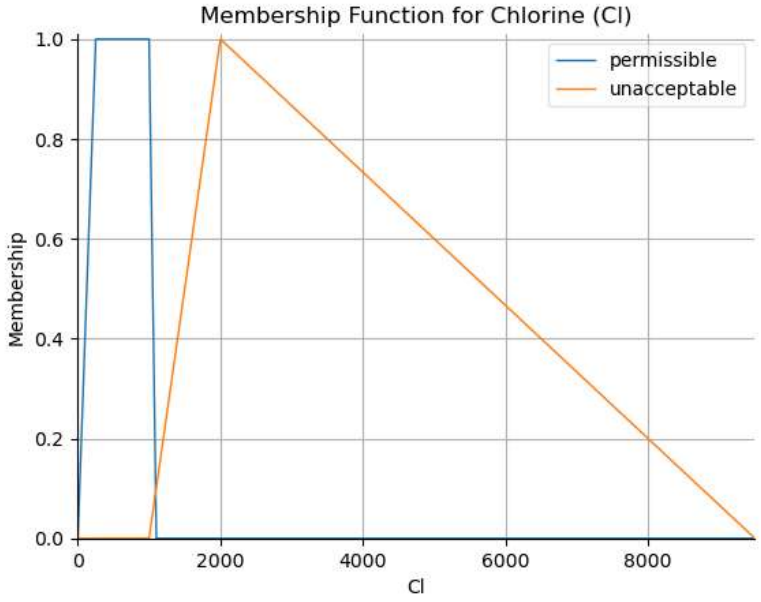


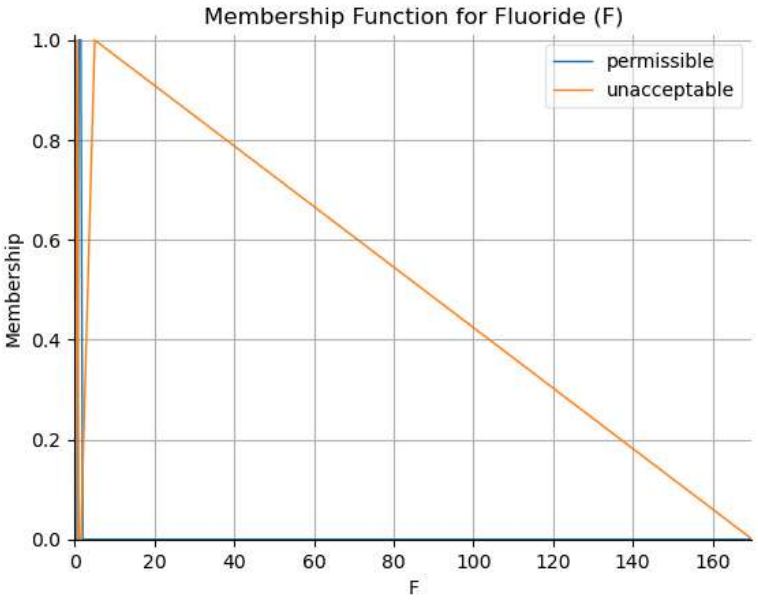
In [26]:
```
# 4. Membership functions for output (WQI)
wqi['excellent'] = fuzz.trimf(wqi.universe, [0, 15, 30])
wqi['good'] = fuzz.trimf(wqi.universe, [25, 35, 55])
wqi['bad'] = fuzz.trimf(wqi.universe, [50, 65, 80])
wqi['very_bad'] = fuzz.trimf(wqi.universe, [75, 90, 105])
wqi['unfit'] = fuzz.trimf(wqi.universe, [100, 120, 200])
```

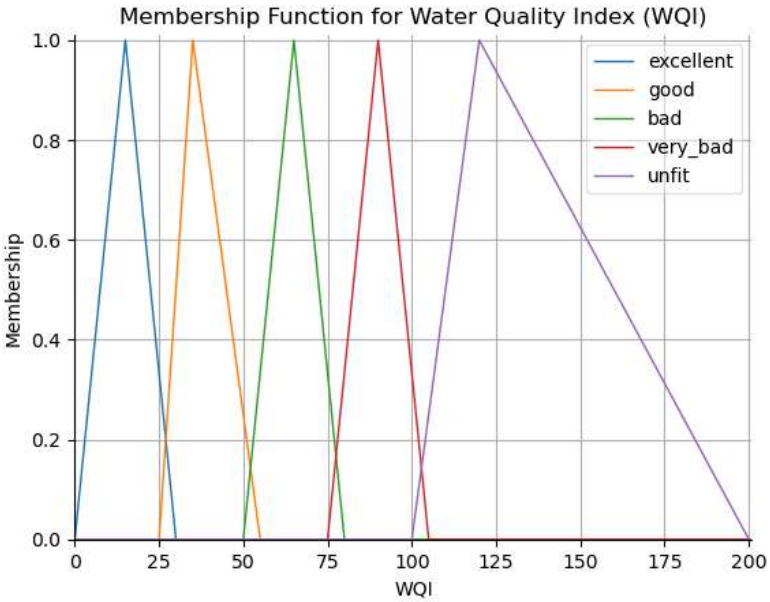
In [27]:
```
# 5. Visualising all the membership functions

# List of variables and their corresponding titles
variables = [
    (ph, 'Membership Function for pH'),
    (th, 'Membership Function for Total Hardness (TH)'),
    (cl, 'Membership Function for Chlorine (Cl)'),
    (f, 'Membership Function for Fluoride (F)'),
    (wqi, 'Membership Function for Water Quality Index (WQI)')
]

# Loop through each variable and plot
for var, title in variables:
    var.view()
    plt.title(title)
    plt.grid()
    plt.show()
```


Membership Function for pH
permissible
unacceptable
Membership
pH
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
12


Membership Function for Total Hardness (TH)
permissible
unacceptable
Membership
TH
0.0
0.2
0.4
0.6
0.8
1.0
0
1000
2000
3000
4000
5000
6000
7000
8000


Membership Function for Chlorine (Cl)
permissible
unacceptable
Membership
Cl
0.0
0.2
0.4
0.6
0.8
1.0
0
2000
4000
6000
8000





Rules used for classifying WQI category

In [29]:
```
# Displaying the image for the rule base used and alloted WQI category
display(Image(filename=r"D:\Data Science\Projects\3. Fuzzy Logic for Water Quality Assessment Project\WQI_category_rules.png"))
```

`unacceptable` count	`permissible` count	WQI Label
≥2	Any	`unfit`
1	≥2	`very_bad`
1	<2	`bad`
0	4	`excellent`
0	≤3	`bad`

In [30]:

```python
# 6. Generate fuzzy rules (based on number of 'permissible' and 'unacceptable' inputs)
from itertools import product

labels = ['permissible', 'unacceptable']
rule_list = []

# Generate combinations for 4 parameters (pH, TH, Cl, F)
combinations = list(product(labels, repeat=4))

for ph_lbl, th_lbl, cl_lbl, f_lbl in combinations:
    status = [ph_lbl, th_lbl, cl_lbl, f_lbl]
    unacc_count = status.count('unacceptable')
    perm_count = status.count('permissible')

    # Final rule logic
    if unacc_count >= 2:
        wqi_lbl = 'unfit'
    elif unacc_count == 1 and perm_count >= 2:
        wqi_lbl = 'very_bad'
    elif unacc_count == 1 and perm_count < 2:
        wqi_lbl = 'bad'
    elif unacc_count == 0 and perm_count == 4:
        wqi_lbl = 'excellent'
    elif unacc_count == 0 and perm_count <= 3:
        wqi_lbl = 'bad'
    else:
        wqi_lbl = 'bad'  # Fallback, though all paths are covered

    # Create fuzzy rule
    rule = ctrl.Rule(
        antecedent=(ph[ph_lbl] & th[th_lbl] & cl[cl_lbl] & f[f_lbl]),
        consequent=wqi[wqi_lbl],
        label=f'Rule_{ph_lbl}_{th_lbl}_{cl_lbl}_{f_lbl}'
    )
    rule_list.append(rule)


# 7. Build the control system and simulation
wqi_ctrl = ctrl.ControlSystem(rule_list)
wqi_sim = ctrl.ControlSystemSimulation(wqi_ctrl)

# 8. Function to compute WQI (Defuzzification)
def compute_wqi(row):
    wqi_sim.reset()
    wqi_sim.input['pH'] = row['pH']
    wqi_sim.input['TH'] = row['TH']
    wqi_sim.input['Cl'] = row['Cl']
    wqi_sim.input['F'] = row['F']
    try:
        wqi_sim.compute()
        return wqi_sim.output['WQI']
    except:
        return np.nan

# 8. Applying the function to our DataFrame
df['WQI'] = df.apply(compute_wqi, axis=1)

# 9. Categorize WQI values
def wqi_category(value):
    if pd.isna(value):
        return 'Unknown'
    elif value <= 25:
        return 'Excellent'
    elif value <= 50:
        return 'Good'
    elif value <= 75:
        return 'Bad'
    elif value <= 100:
        return 'Very Bad'
    else:
        return 'Unfit'

df['WQI_Category'] = df['WQI'].apply(wqi_category)

# 10. Display result
print(df[['pH', 'TH', 'Cl', 'F', 'WQI', 'WQI_Category']])
```

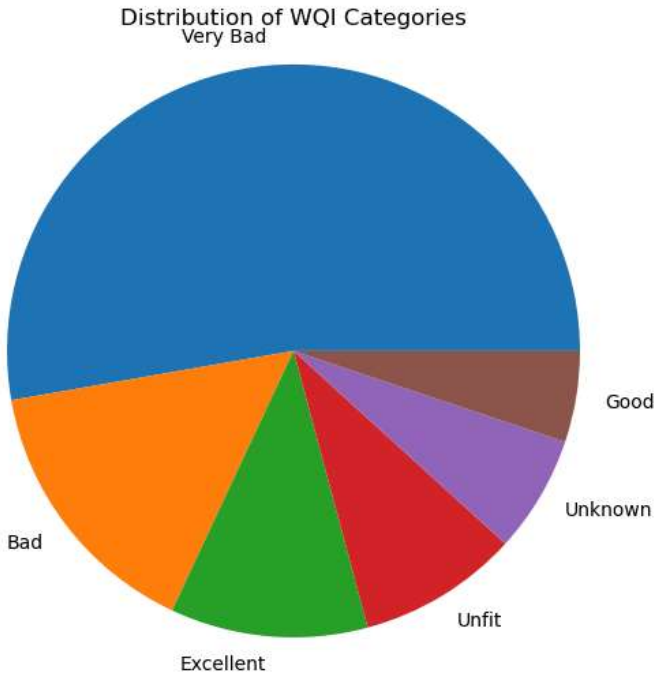
```
         pH      TH      Cl    F          WQI WQI_Category
0       7.9   465.0   120.5  0.5    90.000000     Very Bad
1       7.6   510.0   191.4  0.2    90.000000     Very Bad
2       7.4  1900.0  1201.8  0.1   147.111111        Unfit
3       8.0   425.0   102.8  0.3    90.000000     Very Bad
4       7.8   290.0    24.8  0.2    90.000000     Very Bad
...     ...     ...     ...  ...          ...          ...
13595   7.7   500.0  1077.7  2.0   129.045628        Unfit
13596   7.7  2900.0  5672.0  0.9   143.218423        Unfit
13597   8.1   380.0   567.2  3.8    90.000000     Very Bad
13598   7.6   640.0   581.4  0.8   111.745070        Unfit
13599   8.2   380.0  1843.4  1.2    90.000000     Very Bad

[13600 rows x 6 columns]
```

Visualization and Inference of Results

In [32]:
```
category_counts = df['WQI_Category'].value_counts()
category_counts
```

Out[32]:
```
WQI_Category
Very Bad     7175
Bad          2078
Excellent    1515
Unfit        1234
Unknown       898
Good          700
Name: count, dtype: int64
```

In [33]:
```
# Plotting pie chart of distribution of WQI categories
plt.figure(figsize=(8, 6))
plt.pie(category_counts, labels=category_counts.index)
plt.title('Distribution of WQI Categories')
plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
plt.show()
```



Calculating and displaying average WQI of each district

In [35]:
```
new_df = pd.concat([df_org[['STATE', 'DISTRICT']], df[['WQI', 'WQI_Category']]], axis=1)
new_df.sample(5)
```

Out[35]:

	STATE	DISTRICT	WQI	WQI_Category
13139	RAJASTHAN	JHALAWAR	15.000000	Excellent
4375	MADHYA PRADESH	RAISEN	90.000000	Very Bad
12073	HARYANA	FARIDABAD	47.510179	Good
5629	U.P	KUSHINAGAR	NaN	Unknown
7406	TAMIL NADU	VIRUDHUNAGAR	25.209902	Good

In [36]:
```
district_wqi = new_df.groupby('DISTRICT')['WQI'].mean().reset_index()
district_wqi.columns = ['District', 'Average_WQI']
print(district_wqi)
```

```
             District  Average_WQI
0            ADILABAD    62.526720
1          AGAR MALWA    79.296547
2               AGRA     75.213784
3           AHMEDABAD    99.452542
4          AHMEDNAGAR    89.526042
..                ...          ...
569  WEST KHASI HILLS    90.000000
570      WEST TRIPURA   104.805581
571            YADGIR    68.650935
572       YAMUNANAGAR    85.541603
573          YAVATMAL    68.446700

[574 rows x 2 columns]
```

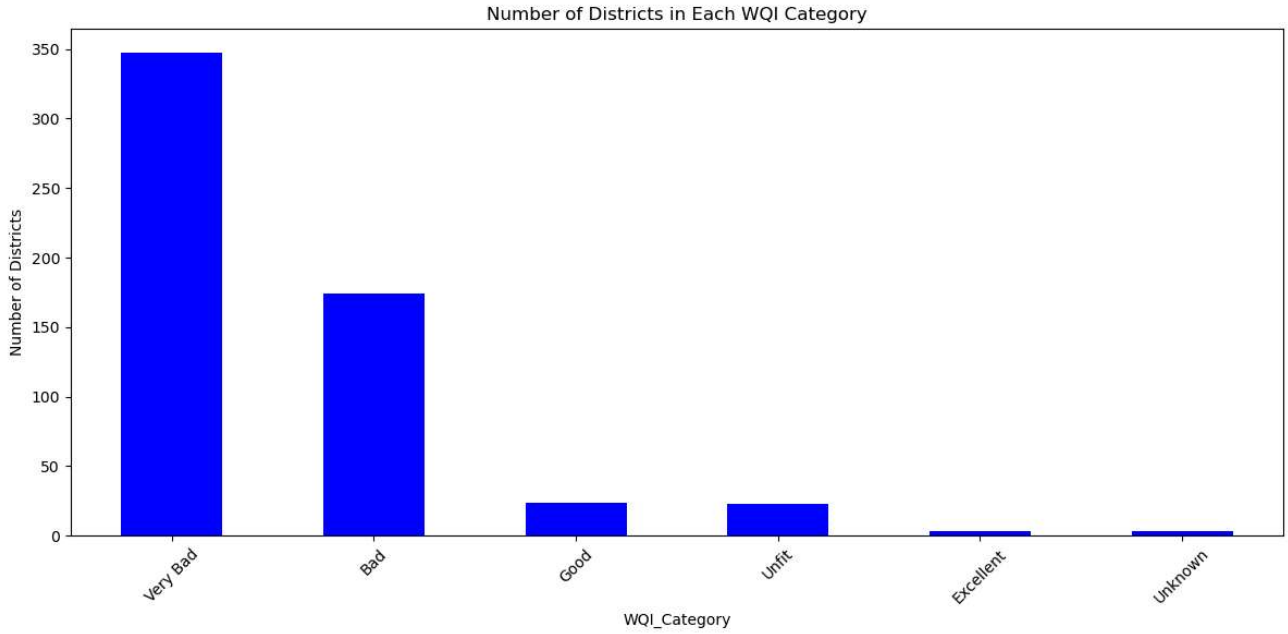
In [37]:
```
district_wqi['WQI_Category'] = district_wqi['Average_WQI'].apply(wqi_category)
print(district_wqi.sample(10))
```

```
           District  Average_WQI WQI_Category
423  PURBA MEDINIPUR    87.500000     Very Bad
114      CHITRADURGA    71.268457          Bad
91             BOTAD    88.480390     Very Bad
311             KOTA    63.747898          Bad
221            HOJAI          NaN      Unknown
43       BALODABAZAR    78.323371     Very Bad
98         BURHANPUR    83.714330     Very Bad
464           SANGLI    85.538887     Very Bad
242         JAMNAGAR   101.435557        Unfit
144       DHARMAPURI    78.441759     Very Bad
```

In [38]:
```
dist_cat_counts = district_wqi['WQI_Category'].value_counts()
dist_cat_counts
```

Out[38]:
```
WQI_Category
Very Bad     347
Bad          174
Good          24
Unfit         23
Excellent      3
Unknown        3
Name: count, dtype: int64
```

In [39]:
```
plt.figure(figsize=(12, 6))
dist_cat_counts.plot(kind='bar', color='blue')
plt.ylabel('Number of Districts')
plt.title('Number of Districts in Each WQI Category')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



Ranking States based on average WQI

In [41]:
```
# Group by state and calculate average WQI
state_avg_wqi = new_df.groupby('STATE')['WQI'].mean().reset_index()
state_avg_wqi.columns = ['State', 'Average_WQI']

# Sort in increasing order (lower WQI = better)
state_avg_wqi = state_avg_wqi.sort_values(by='Average_WQI', ascending=True).reset_index(drop=True)

# Assign ranks
state_avg_wqi['Rank'] = state_avg_wqi.index + 1

# Display without index
print(state_avg_wqi.to_string(index=False))
```

```
          State  Average_WQI  Rank
          BIHAR    63.356604     1
      TELANGANA    64.639441     2
     PONDICHERRY    67.610381     3
      TAMIL NADU    71.997137     4
      JHARKHAND    74.100432     5
    MAHARASHTRA    74.104834     6
   MADHYA PRADESH    74.644045     7
   ANDHRA PRADESH    74.700772     8
            U.P    75.226154     9
     CHHATTISGARH    75.557303    10
        KARNATAKA    77.847651    11
        RAJASTHAN    79.644906    12
           PUNJAB    81.133326    13
           ODISHA    82.915814    14
  DADRA AND NAGAR    83.193478    15
ARUNACHAL PRADESH    86.875000    16
      UTTARAKHAND    87.368421    17
            ASSAM    88.590581    18
  JAMMU & KASHMIR    88.933581    19
          TRIPURA    89.811994    20
      WEST BENGAL    89.919057    21
         NAGALAND    90.000000    22
    CHANDIGARH UT    90.000000    23
          HARYANA    90.333180    24
          GUJARAT    91.261179    25
        MEGHALAYA    91.853109    26
           KERALA    94.022272    27
```

In [ ]: