IoT-Based Water Supply Management System

Design an IoT-Based Water Supply Management System to optimize water distribution and consumption in urban and rural settings. This software solution should enable real-time monitoring and management of water supply networks, utilizing IoT sensors to collect data on water flow, pressure, quality, and consumption patterns. The system must feature a user-friendly interface that displays critical metrics and visualizations through interactive dashboards, allowing water utility managers to track system performance and identify leaks or inefficiencies. Additionally, the platform should incorporate predictive analytics to forecast water demand based on historical data and seasonal trends, enabling proactive resource allocation and maintenance scheduling. The challenge is to develop robust algorithms for data analysis and integrate communication protocols that ensure seamless connectivity among all IoT devices. By providing actionable insights and improving operational efficiency, this solution will promote sustainable water management practices, enhance service delivery, and contribute to conservation efforts in the face of growing water scarcity.

The statement outlines the creation of an IoT-Based Water Supply Management System designed to enhance water distribution and consumption practices in both urban and rural environments. Here is a detailed breakdown of the key components and objectives of this solution:

Optimize Water Distribution and Consumption: Aim to manage water resources effectively, ensuring that both urban and rural areas receive adequate water supply while minimizing waste and overconsumption.

In [1]:
```
#import the necessary modules

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

In [2]:
```
#Loading the Dataset
df = pd.read_csv("water_supply_dataset.csv")
df.head()
```

Out[2]:

	Timestamp	Sensor_ID	Water_Flow_Rate	Water_Pressure	Water_Temperature	W
0	2024-10-18 08:06:24.179489	1936	123.012581	458742.877621	11.436859	
1	2024-10-18 08:05:24.179500	1740	61.706821	212911.647084	27.884570	
2	2024-10-18 08:04:24.179502	1416	59.250224	163731.693406	21.065049	
3	2024-10-18 08:03:24.179503	1441	81.023858	457735.726387	22.790899	
4	2024-10-18 08:02:24.179505	1163	67.688319	320732.978098	14.563713	

In [ ]:

In [3]:
```
df.shape
```

Out[3]: (5000, 20)

Dataset Imformation:

1. Timestamp: Date and time of the recorded data.
2. Sensor_ID: Unique ID for each IoT sensor.
3. Water_Flow_Rate: Volume of water passing through a point in liters per minute (L/min).
4. Water_Pressure: Pressure in the pipes in Pascal (Pa).
5. Water_Temperature: Temperature of water in degrees Celsius (°C).
6. Water_Quality: Quality score (1-10) based on turbidity, pH, and contaminants.
7. Water_Level: Water level in meters (m).
8. Leak_Detection: Boolean value indicating whether a leak is detected (1 or 0).
9. Valve_Status: Status of control valves (open/closed).
10. Pump_Status: Status of water pumps (active/inactive).
11. Power_Consumption: Power used by pumps and devices in kilowatt-hours (kWh).
12. Region_ID: Identifier for the geographic region.
13. Demand_Prediction: Predicted water demand in liters.
14. Maintenance_Status: Maintenance required (Yes/No).
15. Daily_Usage: Daily water usage in liters.
16. Pressure_Drop: Drop in pressure from the previous reading in Pa.
17. Pipe_Age: Age of the pipe in years.
18. Alert_Status: Boolean indicating if an alert has been triggered.
19. Predicted_Leak: Machine learning-based prediction of a possible leak (1 or 0).
20. Season: The current season (Winter, Spring, Summer, Fall).

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 20 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Timestamp            5000 non-null   object 
 1   Sensor_ID            5000 non-null   int64  
 2   Water_Flow_Rate      5000 non-null   float64
 3   Water_Pressure       5000 non-null   float64
 4   Water_Temperature    5000 non-null   float64
 5   Water_Quality        5000 non-null   int64  
 6   Water_Level          5000 non-null   float64
 7   Leak_Detection       5000 non-null   int64  
 8   Valve_Status         5000 non-null   object 
 9   Pump_Status          5000 non-null   object 
 10  Power_Consumption    5000 non-null   float64
 11  Region_ID            5000 non-null   int64  
 12  Demand_Prediction    5000 non-null   float64
 13  Maintenance_Status   5000 non-null   object 
 14  Daily_Usage          5000 non-null   float64
 15  Pressure_Drop        5000 non-null   float64
 16  Pipe_Age             5000 non-null   int64  
 17  Alert_Status         5000 non-null   int64  
 18  Predicted_Leak       5000 non-null   int64  
 19  Season               5000 non-null   object 
dtypes: float64(8), int64(7), object(5)
memory usage: 781.4+ KB
```

Data Preprocessing

In [5]:
```
#Handling Missing
df.isna().sum()
```

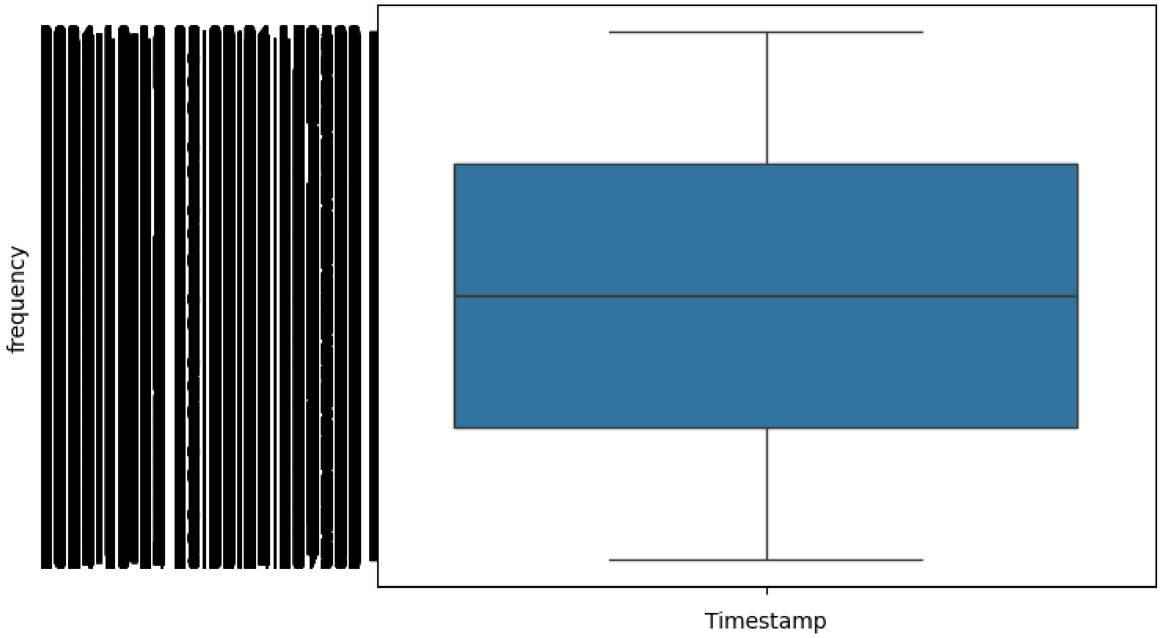
Out[5]:
```
Timestamp             0
Sensor_ID             0
Water_Flow_Rate       0
Water_Pressure        0
Water_Temperature     0
Water_Quality         0
Water_Level           0
Leak_Detection        0
Valve_Status          0
Pump_Status           0
Power_Consumption     0
Region_ID             0
Demand_Prediction     0
Maintenance_Status    0
Daily_Usage           0
Pressure_Drop         0
Pipe_Age              0
Alert_Status          0
Predicted_Leak        0
Season                0
dtype: int64
```

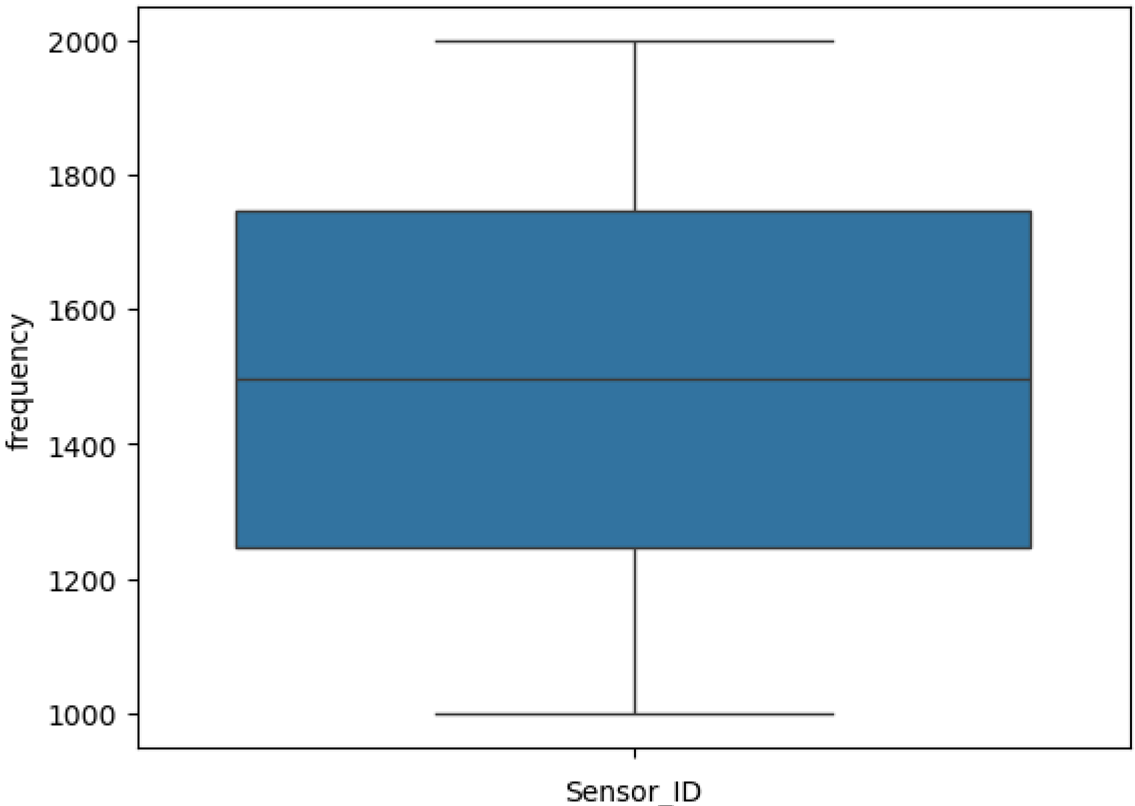
In [6]:
```
#Handling the Duplicate Records
df.duplicated().sum()
```

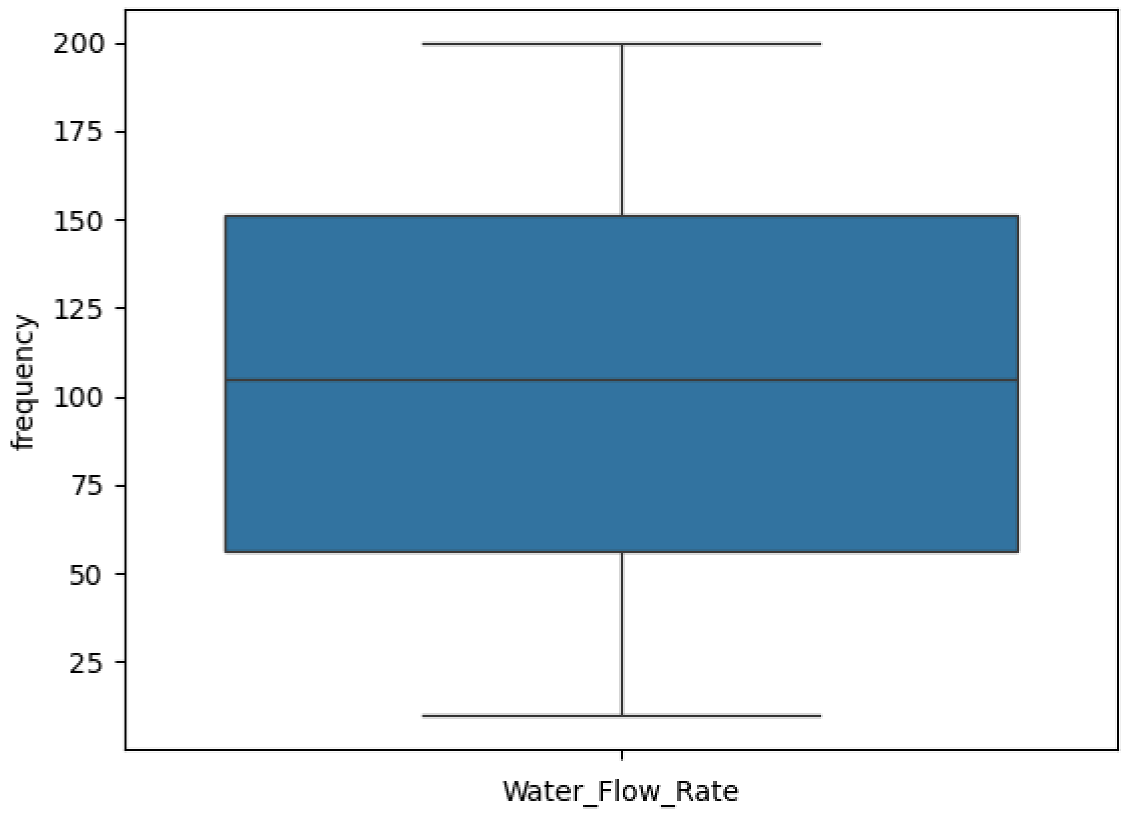
Out[6]: 0

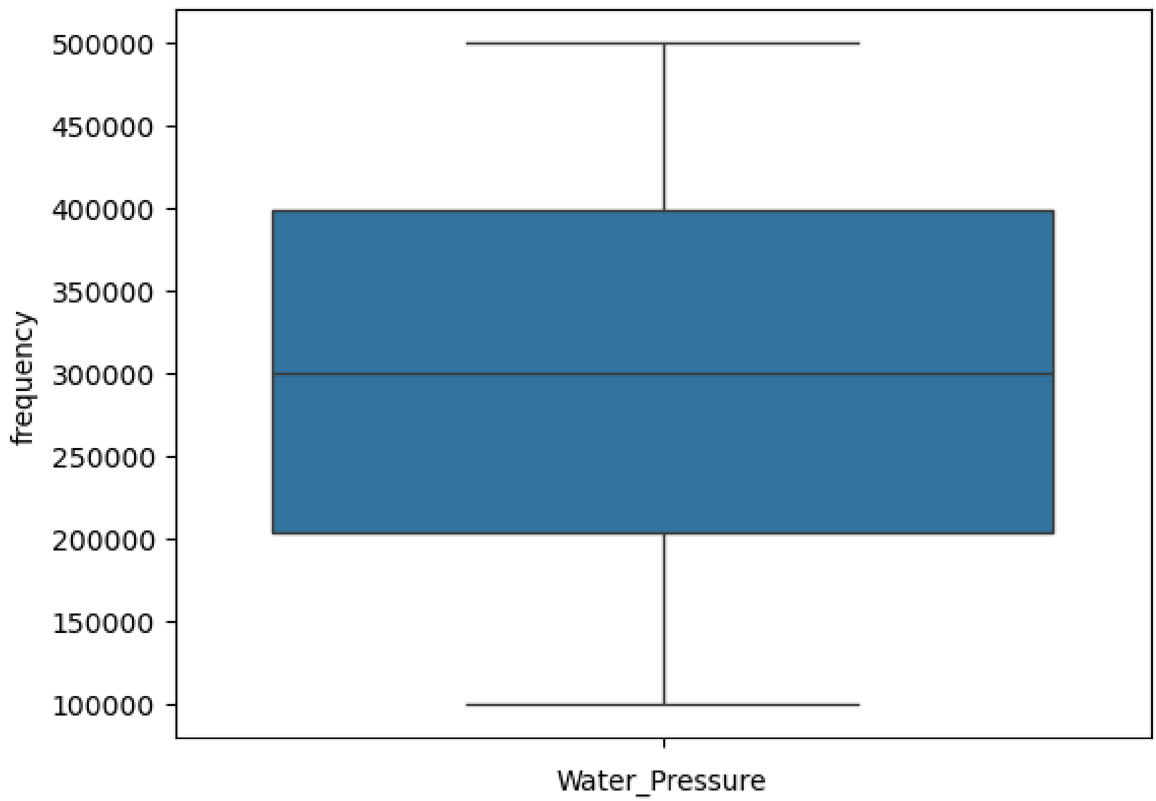
In [7]:
```
# droping the columns 'Predicted_Leak'
df.drop('Predicted_Leak',axis=1,inplace=True)
```

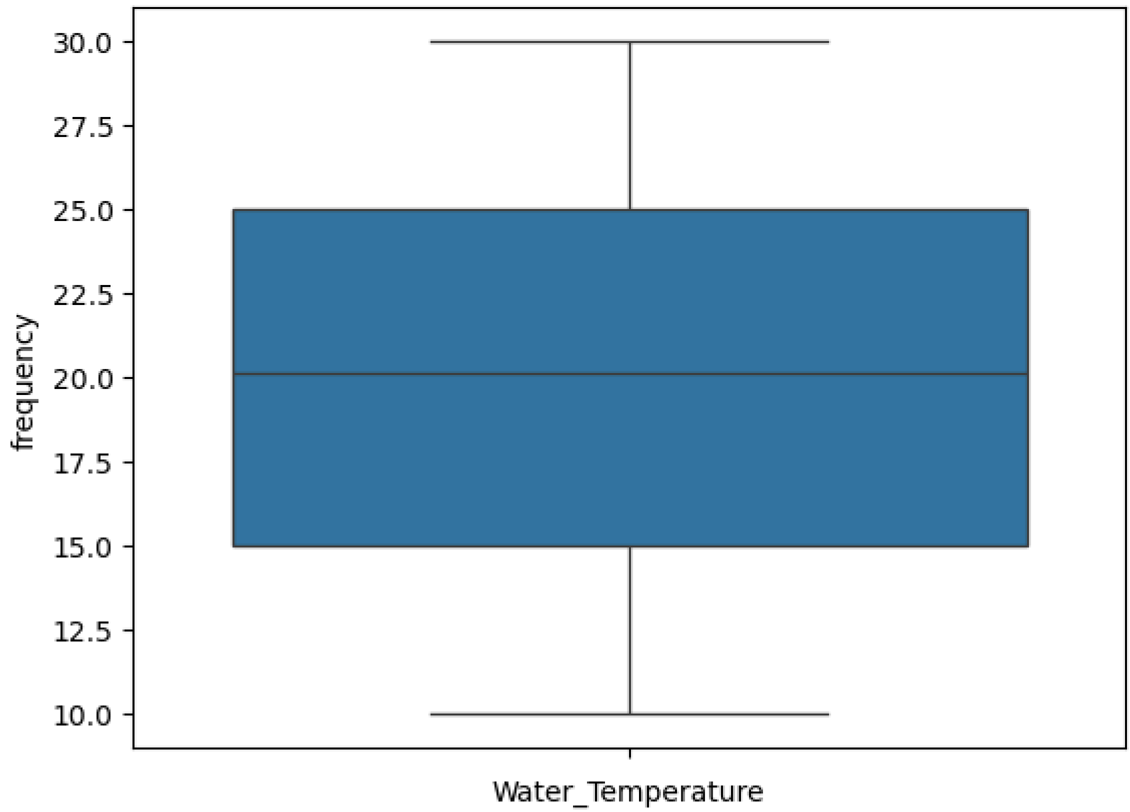
In [8]:
```
#Handling the Outliers
#outlier Analysis
col = df.columns
for i in col:
  if type(i)!='object':
    sns.boxplot(y=df[i])  # Using y= for vertical boxplots
    plt.xlabel(i)  # Set x-label for each subplot
    plt.ylabel('frequency')
    plt.show()
```

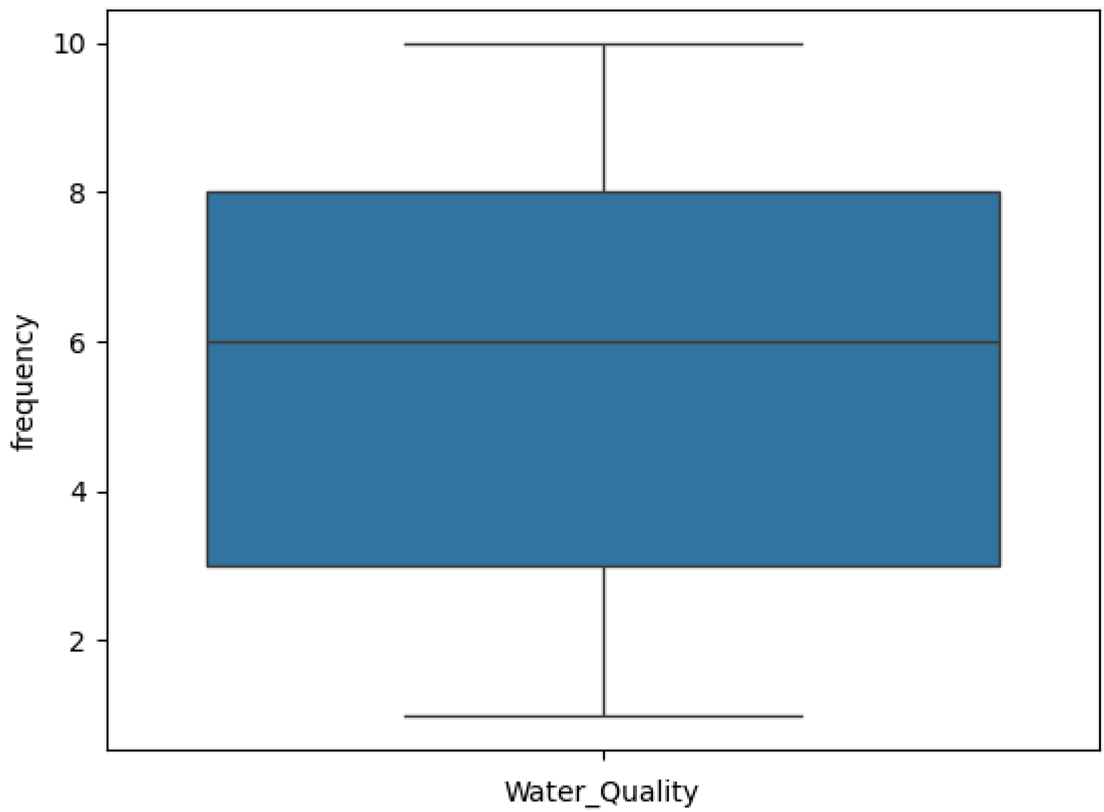


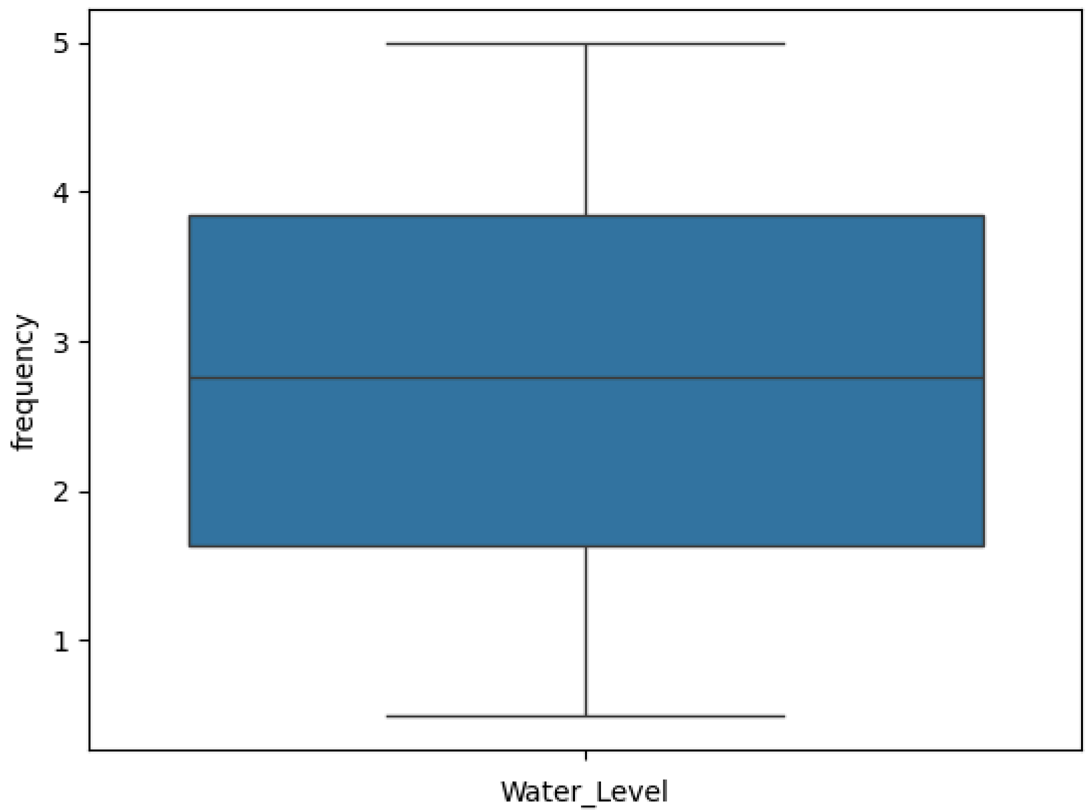



200
175
150
125
100
75
50
25
frequency
Water_Flow_Rate

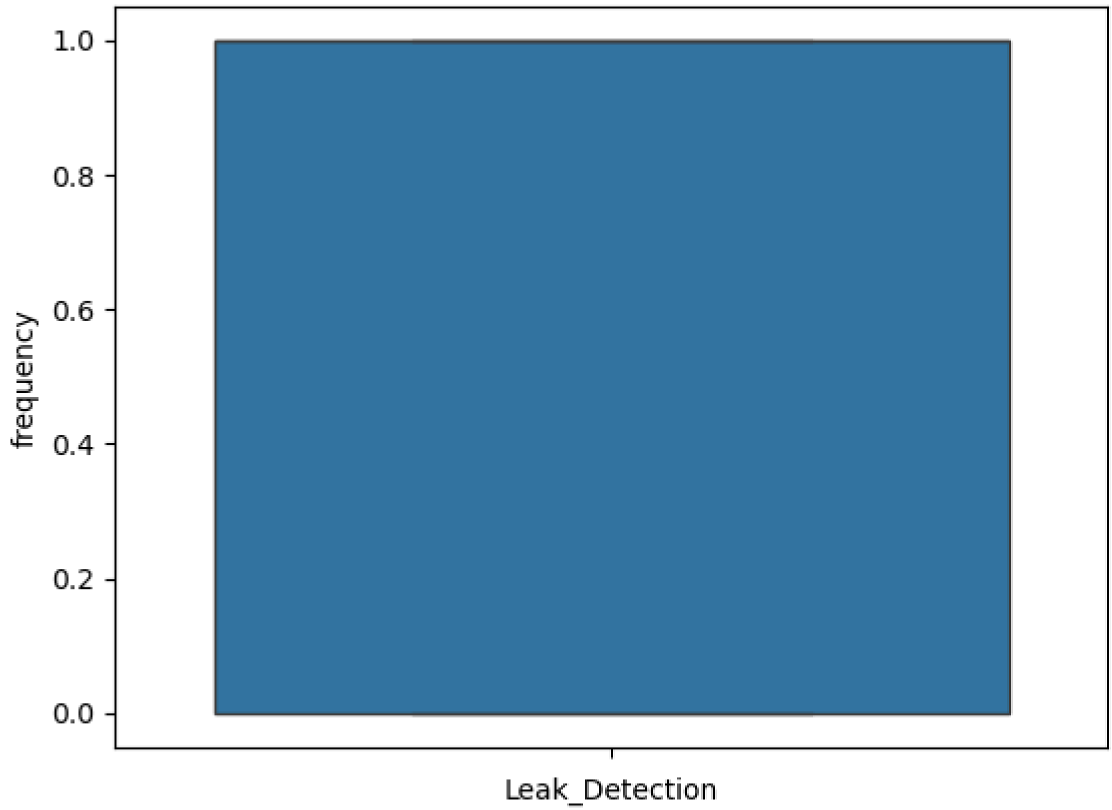

500000
450000
400000
350000
300000
250000
200000
150000
100000
frequency
Water_Pressure


30.0
27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
frequency
Water_Temperature

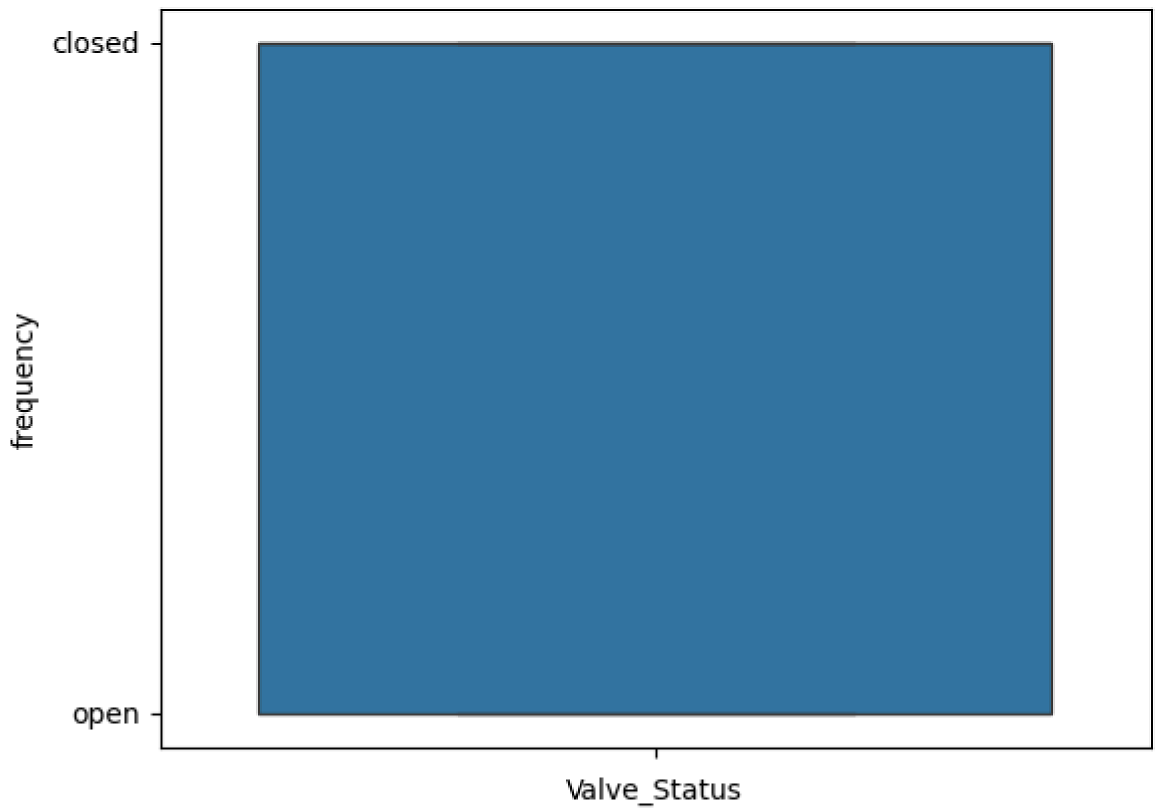

10
8
6
4
2
frequency
Water_Quality

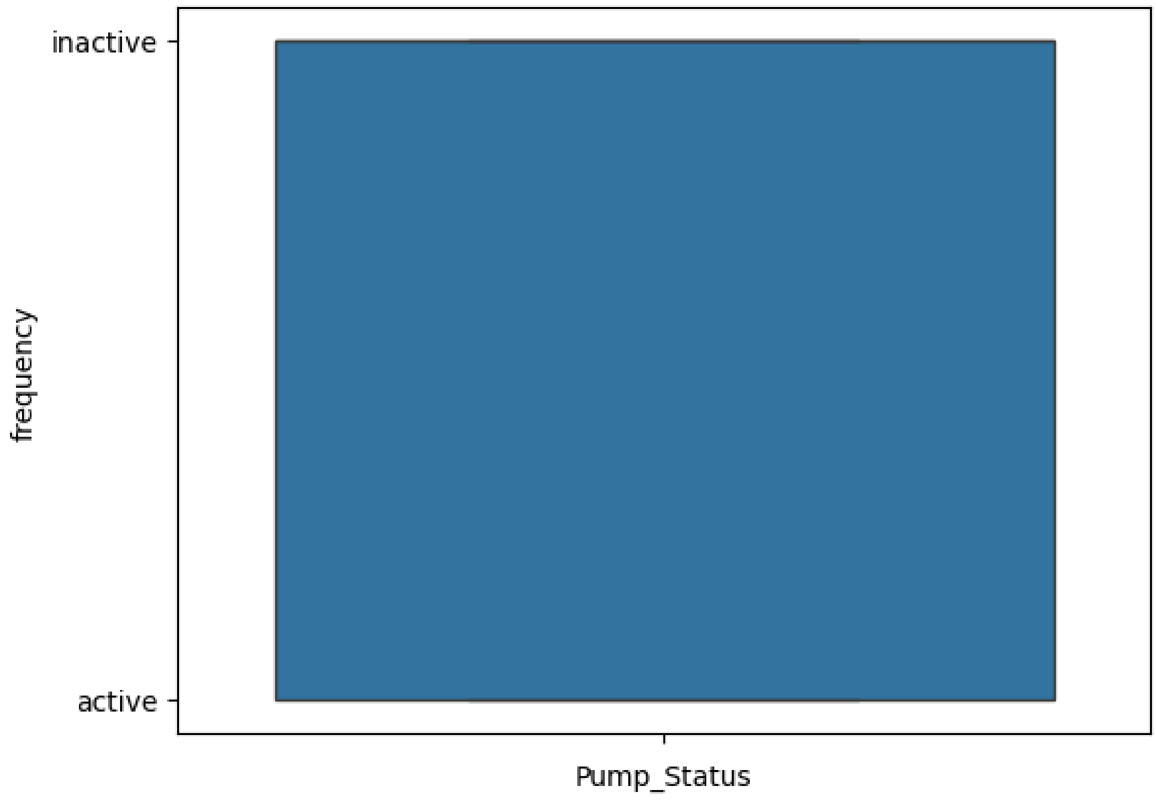


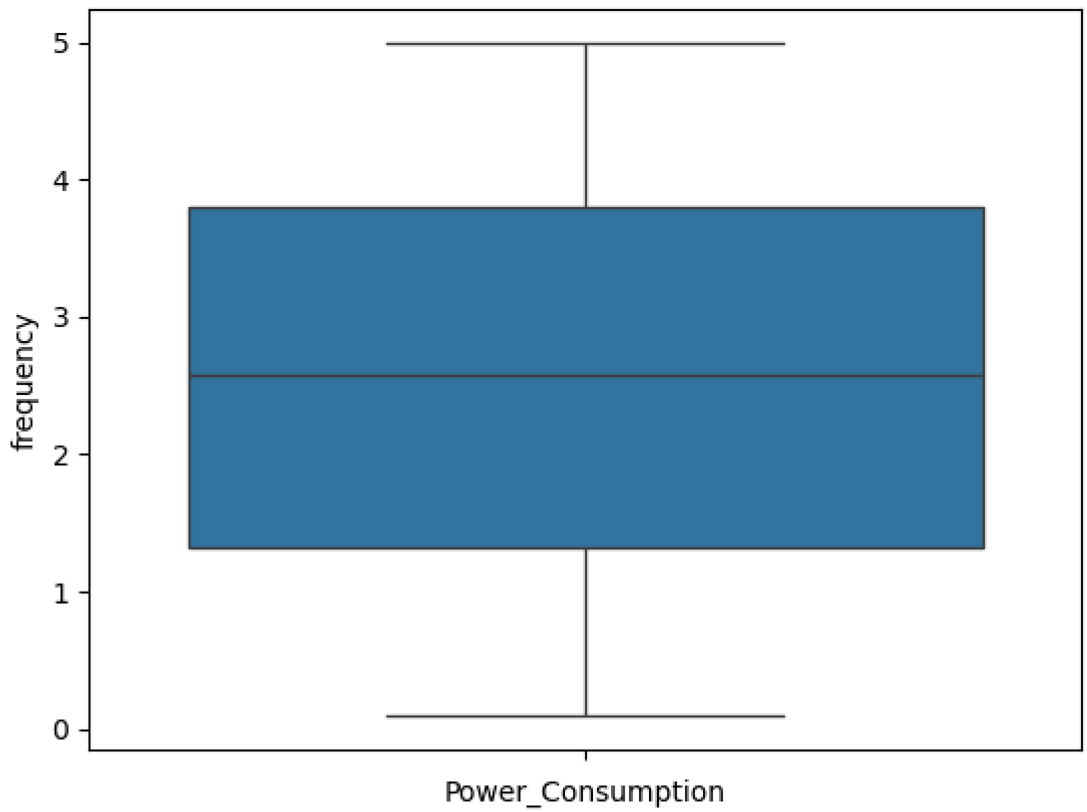
frequency
5
4
3
2
1
Water_Level

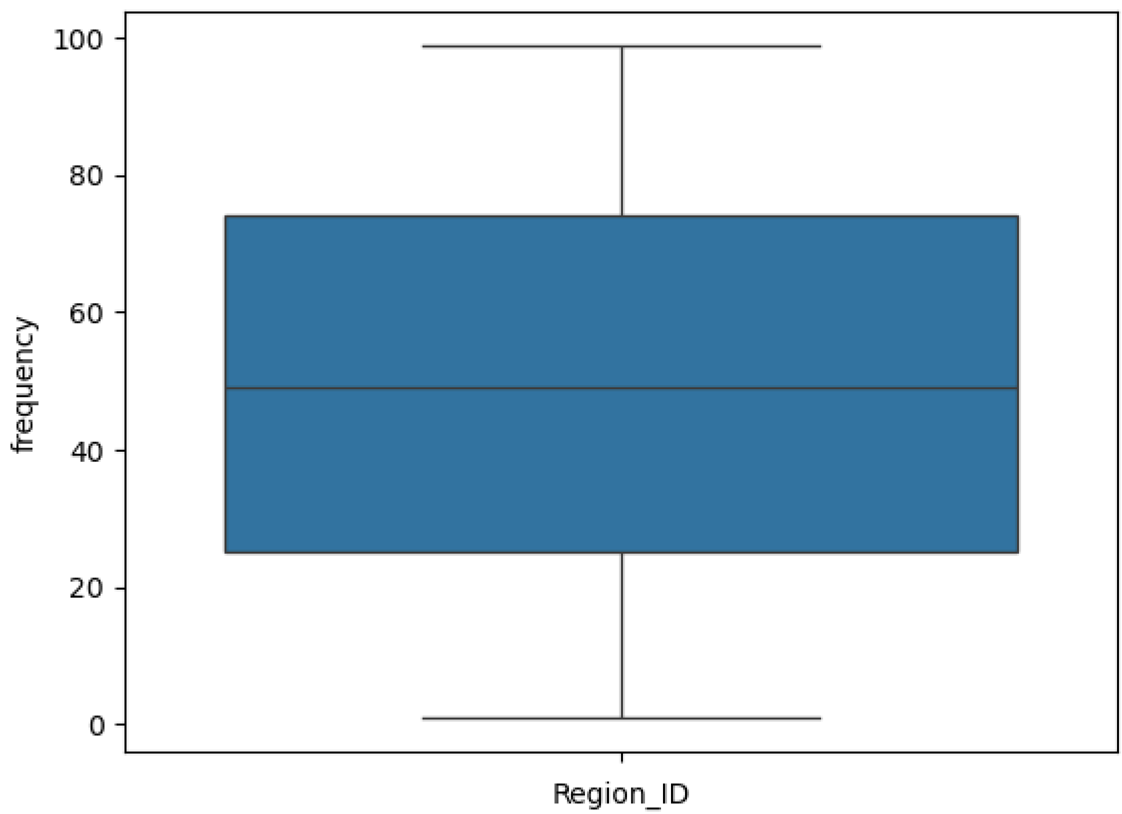


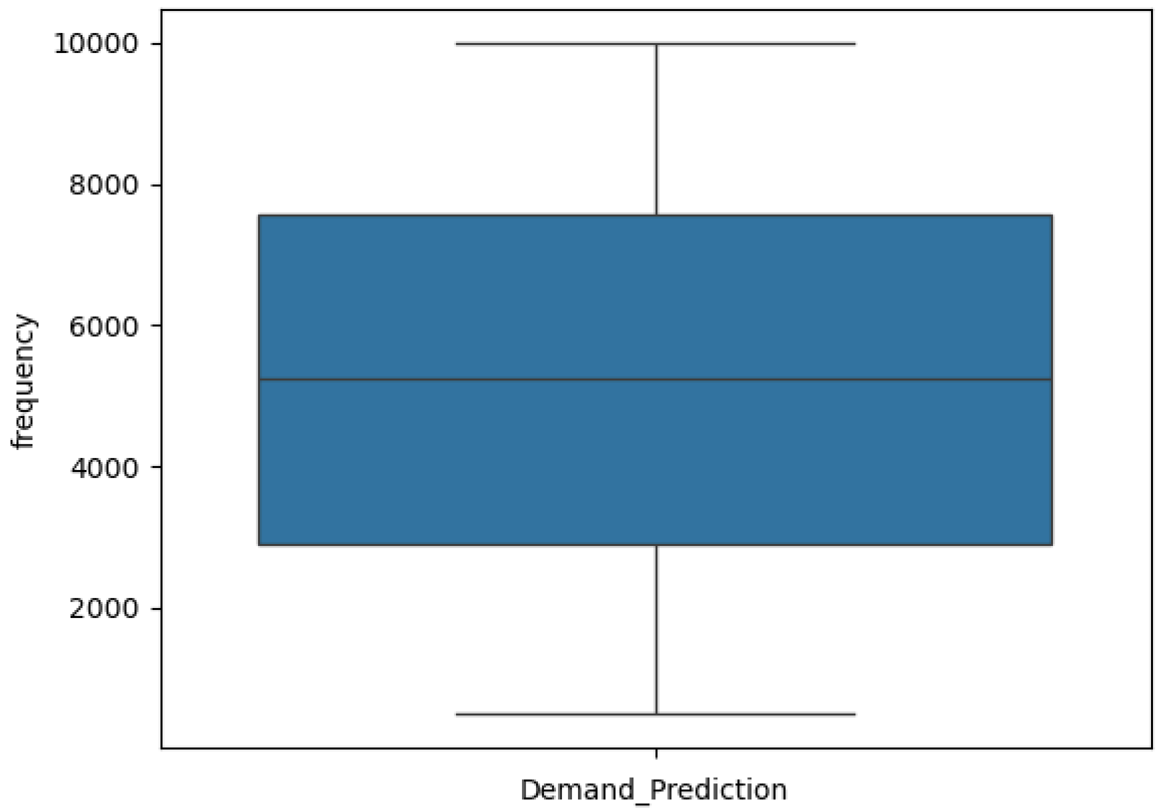
frequency
1.0
0.8
0.6
0.4
0.2
0.0
Leak_Detection

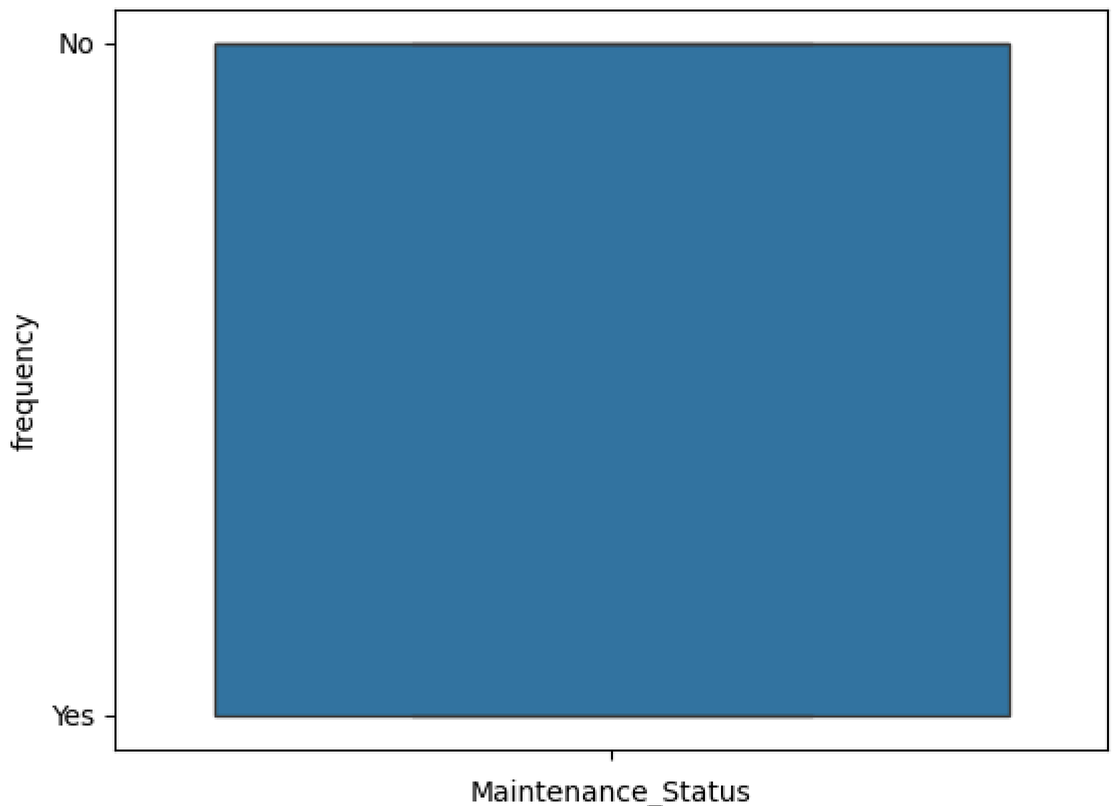

closed
open
frequency
Valve_Status

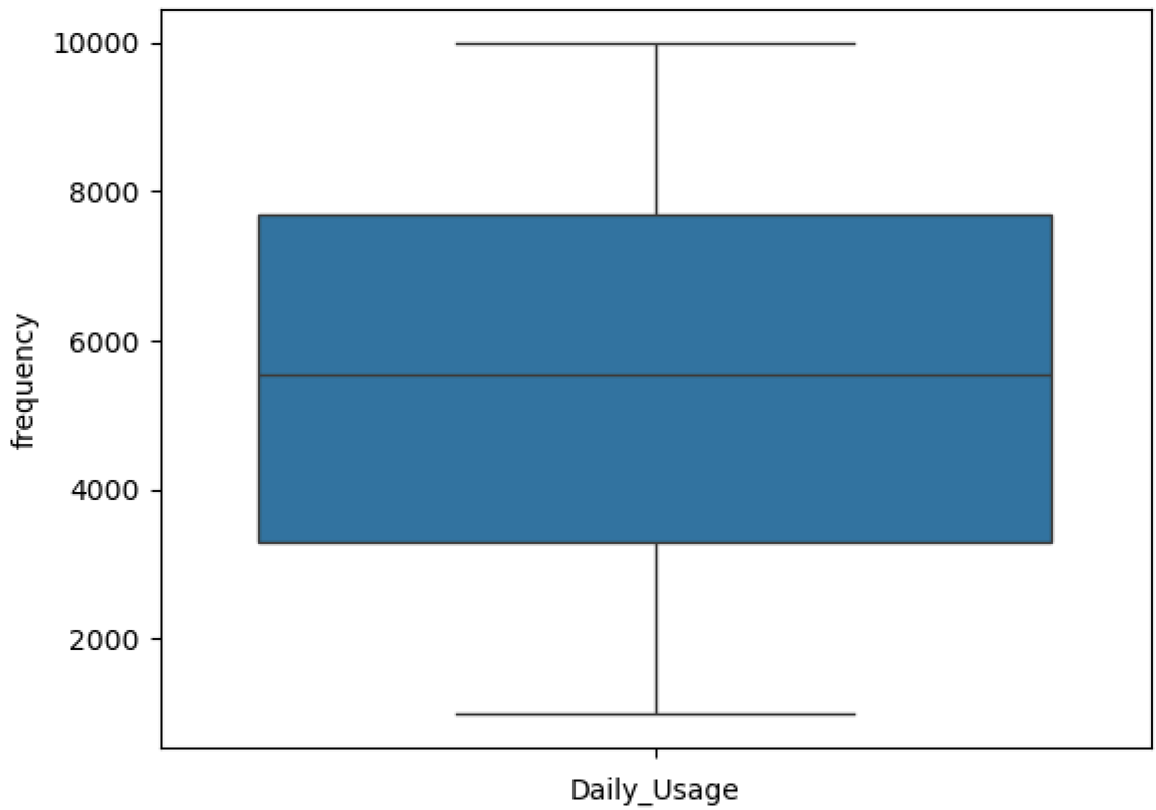

inactive
active
frequency
Pump_Status

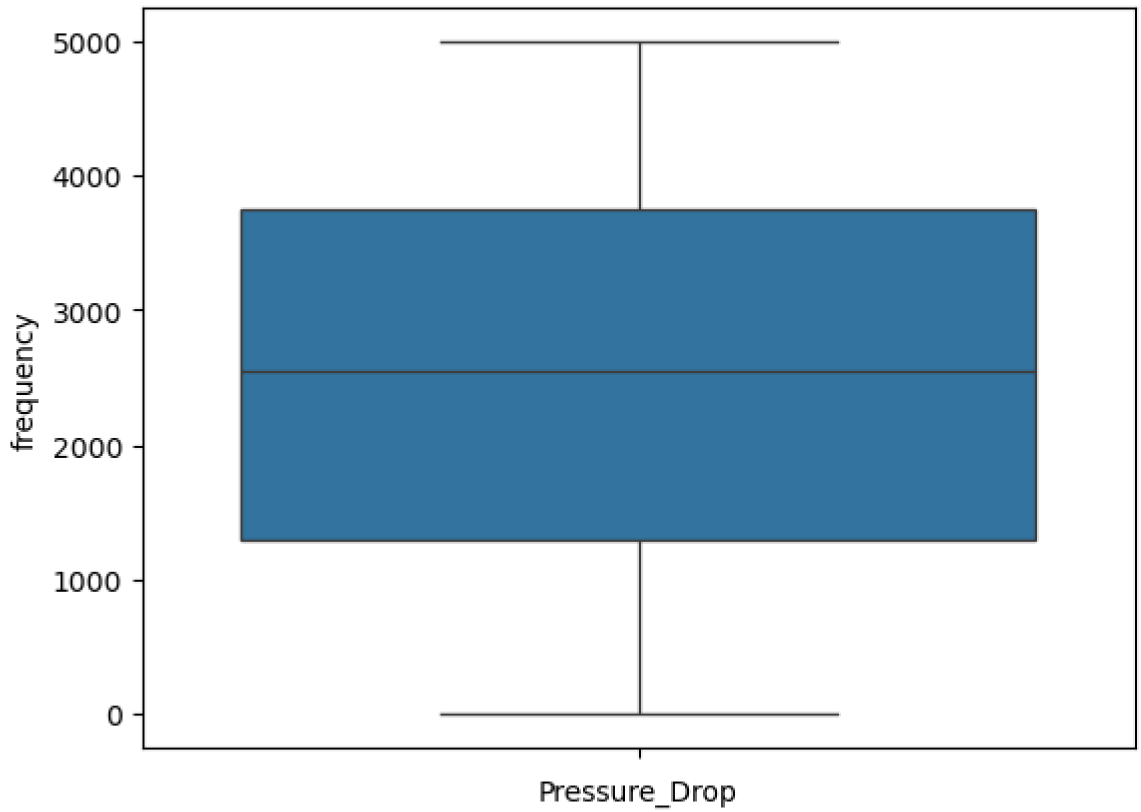

5
4
3
2
1
0
frequency
Power_Consumption

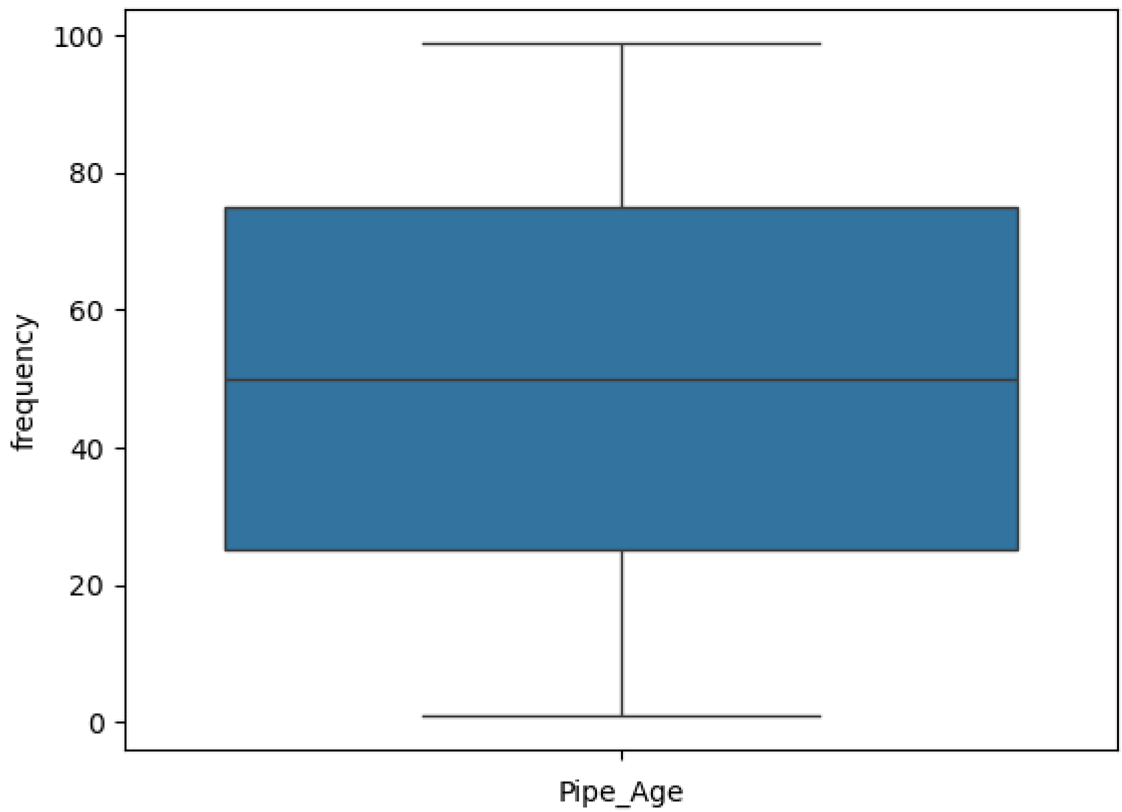

100
80
60
40
20
0
frequency
Region_ID


10000
8000
6000
4000
2000
frequency
Demand_Prediction

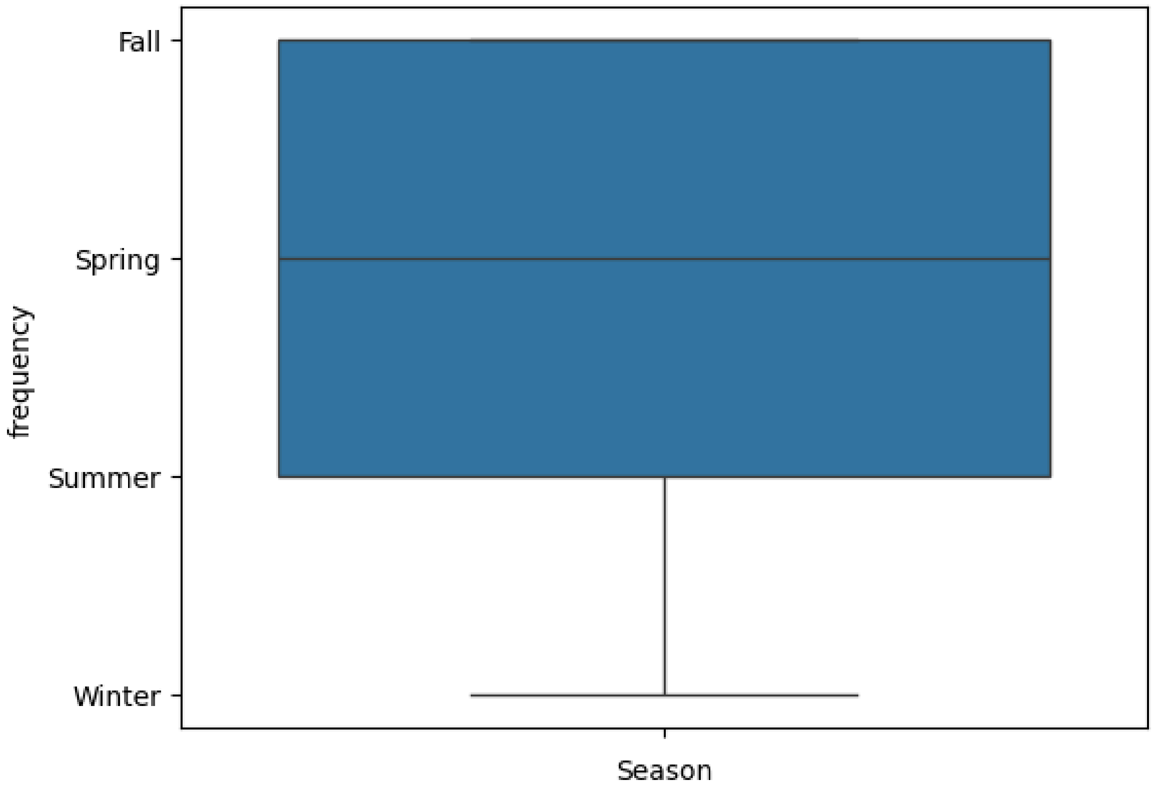

No
Yes
frequency
Maintenance_Status


10000
8000
6000
4000
2000
frequency
Daily_Usage


5000
4000
3000
2000
1000
0
frequency
Pressure_Drop


100
80
60
40
20
0
frequency
Pipe_Age


1.0
0.8
0.6
0.4
0.2
0.0
frequency
Alert_Status



In [9]:
```
# Label Encoding the columns
l = ['Valve_Status','Pump_Status','Maintenance_Status','Alert_Status','Season']
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in l:
  df[i] = le.fit_transform(df[i])
```

In [10]:
```
df.head()
```

Out[10]:

	Timestamp	Sensor_ID	Water_Flow_Rate	Water_Pressure	Water_Temperature	Wa
0	2024-10-18 08:06:24.179489	1936	123.012581	458742.877621	11.436859	
1	2024-10-18 08:05:24.179500	1740	61.706821	212911.647084	27.884570	
2	2024-10-18 08:04:24.179502	1416	59.250224	163731.693406	21.065049	
3	2024-10-18 08:03:24.179503	1441	81.023858	457735.726387	22.790899	
4	2024-10-18 08:02:24.179505	1163	67.688319	320732.978098	14.563713	

In [11]:
```
df['Sensor_ID'].value_counts()
#Inference 1: 994 Sensors are used to collect the data
```

Out[11]:
```
Sensor_ID
1400    14
1171    13
1749    13
1145    11
1501    11
        ..
1586     1
1779     1
1652     1
1839     1
1899     1
Name: count, Length: 994, dtype: int64
```

In [12]:
```
#Finding Average Water Flow Rate
df['Water_Flow_Rate'].mean()
#Inference 2: Average Water Flow Rate is 104 liters/Minute
```

Out[12]:
```
104.73615973879387
```

In [13]:
```
#Maximum and Minimum water flow rate
print(df['Water_Flow_Rate'].max())
print(df['Water_Flow_Rate'].min())
```

```
199.94073540452425
10.052002047714875
```

In [66]:
```
print('Average Water Usage:',df['Daily_Usage'].mean())
print('Minimum Water Usage:',df['Water_Pressure'].min())
print('Maximum Water Usage:',df['Water_Pressure'].max())
```

```
Average Water Usage: 5533.783934313222
Minimum Water Usage: 100084.5505481272
Maximum Water Usage: 499933.1928159776
```

In [14]:
```
#Maximum and Minimum water flow rate at Timestamps
print(df[df['Water_Flow_Rate']==df['Water_Flow_Rate'].max()])
print(df[df['Water_Flow_Rate']==df['Water_Flow_Rate'].min()])
```

```
                     Timestamp  Sensor_ID  Water_Flow_Rate  Water_Pressure  \
1592  2024-10-17 05:34:24.182196       1875       199.940735   136187.515785

      Water_Temperature  Water_Quality  Water_Level  Leak_Detection  \
1592          17.295626              6     3.710123               1

      Valve_Status  Pump_Status  Power_Consumption  Region_ID  \
1592             0            0           3.901785         49

      Demand_Prediction  Maintenance_Status  Daily_Usage  Pressure_Drop  \
1592        3744.746451                   1  3995.033191      1834.4803

      Pipe_Age  Alert_Status  Season
1592        36             1       2
                     Timestamp  Sensor_ID  Water_Flow_Rate  Water_Pressure  \
4037  2024-10-15 12:49:24.185998       1495        10.052002   295997.017295

      Water_Temperature  Water_Quality  Water_Level  Leak_Detection  \
4037          16.838118              4     1.589714               0

      Valve_Status  Pump_Status  Power_Consumption  Region_ID  \
4037             1            0           1.040529         18

      Demand_Prediction  Maintenance_Status  Daily_Usage  Pressure_Drop  \
4037        2750.130365                   0  7038.034608    2895.303904

      Pipe_Age  Alert_Status  Season
4037        92             0       3
```

In [15]:
```python
df['Water_Level'].value_counts()
```

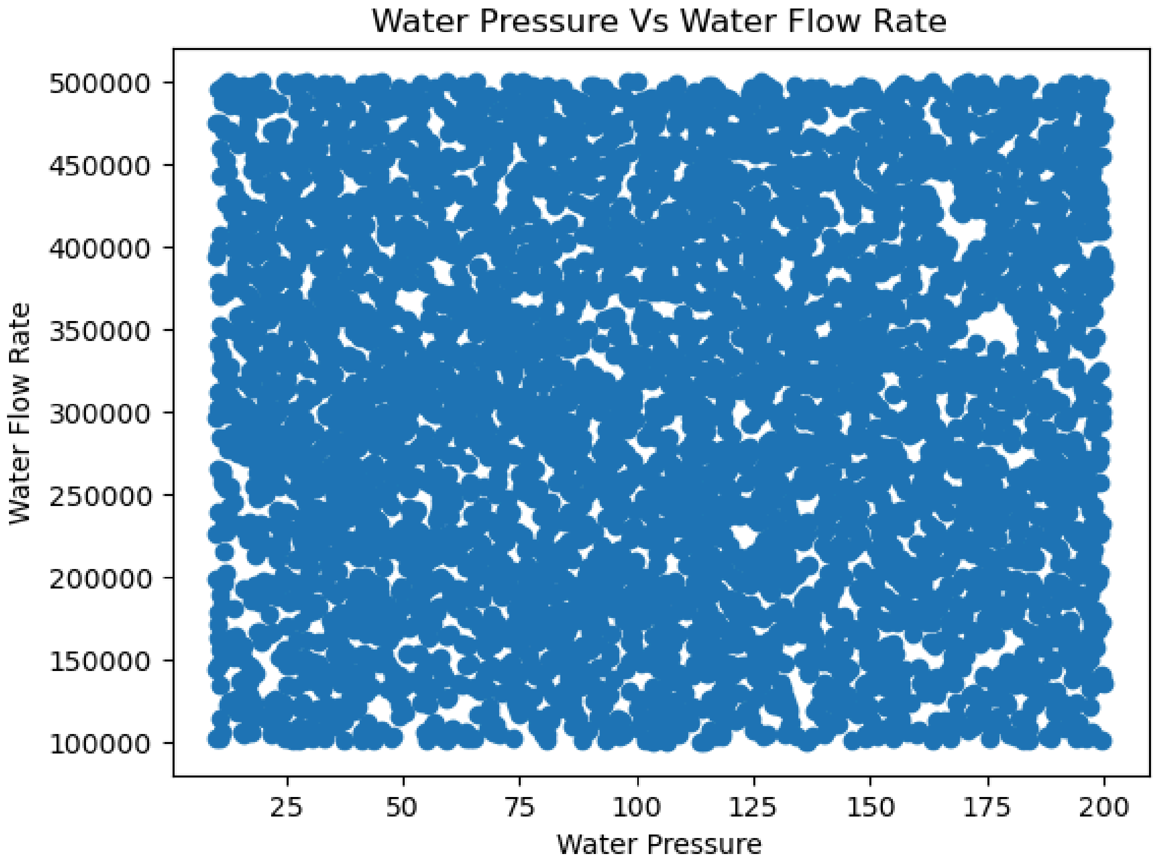
Out[15]:
```
Water_Level
2.629399    1
3.413089    1
2.488904    1
1.467571    1
3.320273    1
           ..
2.713823    1
3.320800    1
2.814505    1
1.352321    1
1.781075    1
Name: count, Length: 5000, dtype: int64
```

In [16]:
```python
print('Average Pressure:',df['Water_Pressure'].mean())
print('Minimum Water Pressure:',df['Water_Pressure'].min())
print('Maximum Water Pressure:',df['Water_Pressure'].max())
```

```
Average Pressure: 300842.52463383926
Minimum Water Pressure: 100050.22268823178
Maximum Water Pressure: 499933.1928159776
```

In [17]:
```python
#Water Pressure Vs Water_Flow_Rate using lineplot
plt.scatter(df['Water_Flow_Rate'],df['Water_Pressure'])
plt.xlabel('Water Pressure')
plt.ylabel('Water Flow Rate')
plt.title('Water Pressure Vs Water Flow Rate')
```

Out[17]:
```
Text(0.5, 1.0, 'Water Pressure Vs Water Flow Rate')
```

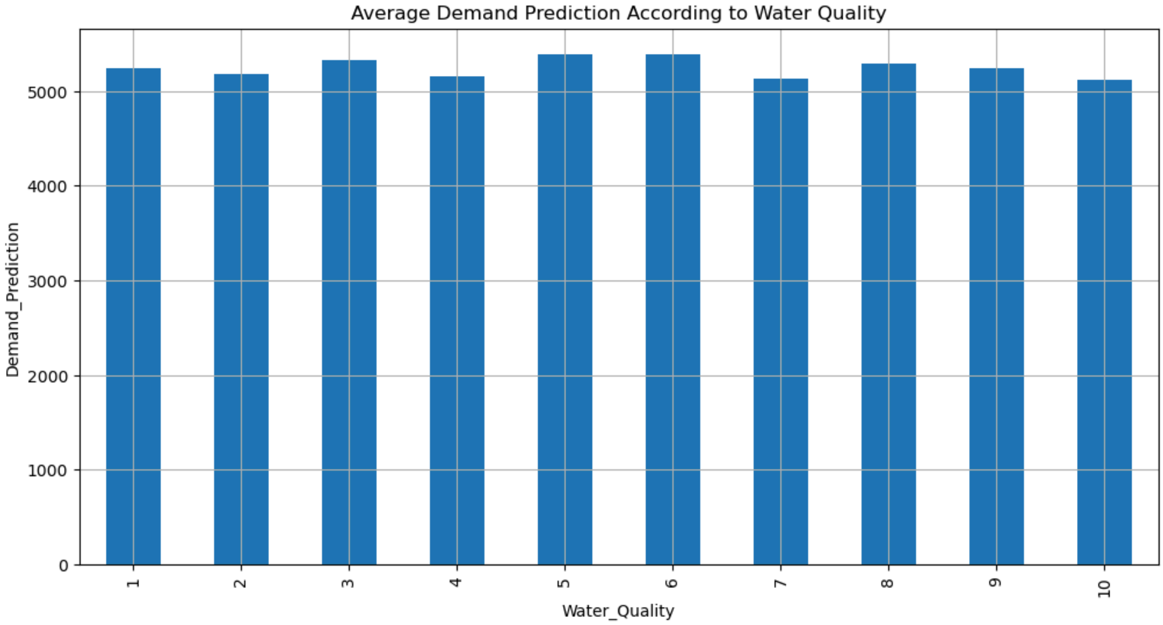


In [18]:
```
fig = px.histogram(df, x="Water_Pressure", color="Water_Flow_Rate", title="<b>Ch
fig.update_layout(width=700, height=500, bargap=0.1)
fig.data[0].marker.color = ('#7fcdff')
fig.data[1].marker.color = ('#326ada')
fig.show()
```

In [19]:

```
store_sales = df.groupby('Water_Quality')['Demand_Prediction'].mean()

# Plot
plt.figure(figsize=(12, 6))
store_sales.plot(kind='bar')
plt.title('Average Demand Prediction According to Water Quality')
plt.xlabel('Water_Quality')
plt.ylabel('Demand_Prediction')
plt.grid()
plt.show()
```



Inference: Demand for Water Quality 5 is High

In [20]:
```
fig = px.histogram(df, x="Leak_Detection", color="Water_Flow_Rate", barmode="grc
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

In [21]:
```
df[df['Leak_Detection']==0]
#2000+ Pipes are Having Water Leakage Problem
```

Out[21]:

	Timestamp	Sensor_ID	Water_Flow_Rate	Water_Pressure	Water_Temperature
1	2024-10-18 08:05:24.179500	1740	61.706821	212911.647084	27.884570
3	2024-10-18 08:03:24.179503	1441	81.023858	457735.726387	22.790899
7	2024-10-18 07:59:24.179510	1113	64.425635	102186.401817	26.757331
9	2024-10-18 07:57:24.179514	1864	82.729540	301860.149515	24.764610
17	2024-10-18 07:49:24.179524	1466	127.500355	248620.056766	15.173109
...	...	...	...	...	...
4989	2024-10-14 20:57:24.187572	1758	32.790702	488479.922608	10.024595
4990	2024-10-14 20:56:24.187577	1664	132.511568	120189.504186	25.751278
4992	2024-10-14 20:54:24.187579	1945	40.091917	291293.217434	19.090008
4997	2024-10-14 20:49:24.187586	1322	150.795986	406852.689983	15.348981
4999	2024-10-14 20:47:24.187589	1359	198.443786	344624.930448	22.842345

2468 rows × 19 columns

In [22]:

```
df[df['Leak_Detection']==0].Water_Flow_Rate.mean()
```

Out[22]: 103.83798446632619

In [23]:

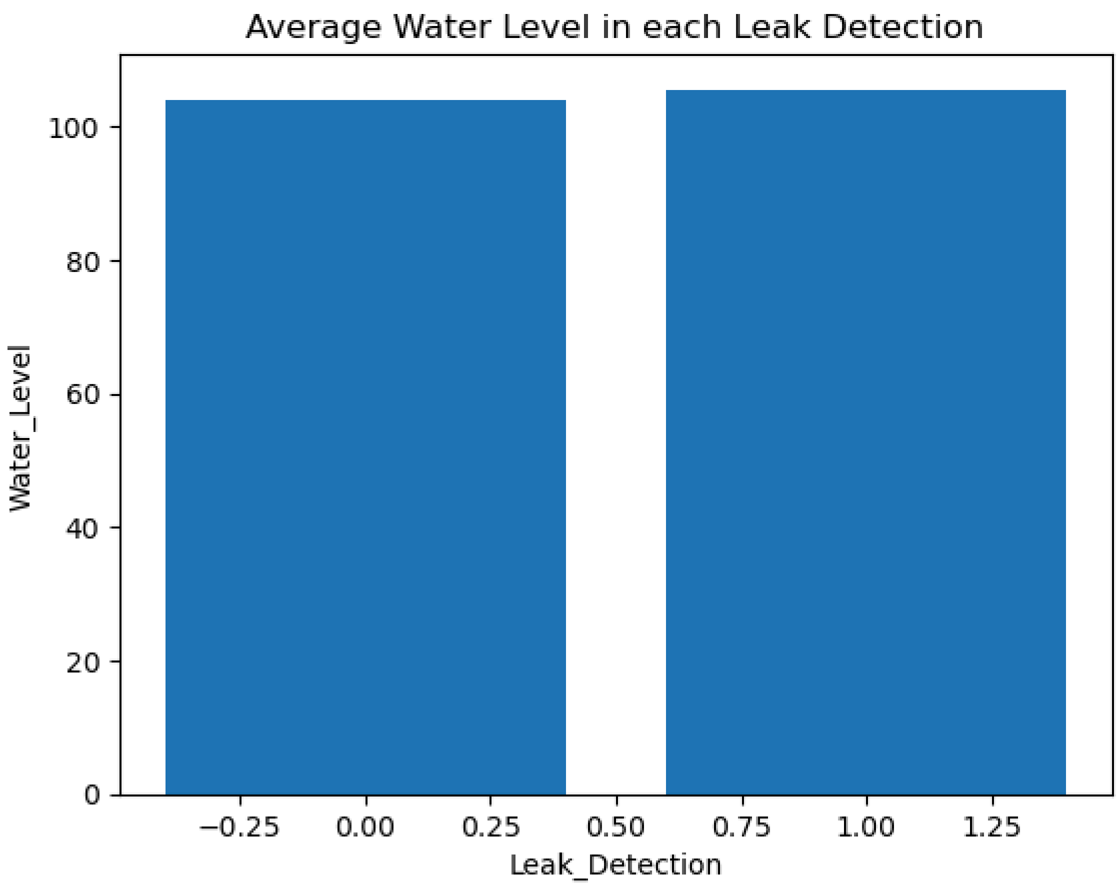
```
df[df['Leak_Detection']==1].Water_Flow_Rate.mean()
```

Out[23]: 105.61163231875051

Inference: Water Flow Rate is High if Leaks are not there when compared to Water Flow rate if leaks exist

In [24]:

```
# Visualizing Water Flow rate According to Leak_Detection
l = [df[df['Leak_Detection']==0].Water_Flow_Rate.mean(),df[df['Leak_Detection']=
plt.bar(range(len(l)),l)
plt.xlabel("Leak_Detection")
plt.ylabel("Water_Level")
plt.title("Average Water Level in each Leak Detection")
plt.show()
```



In [25]:

```
fig = px.histogram(df, x="Leak_Detection", color="Water_Pressure", barmode="grou
fig.update_layout(width=700, height=500, bargap=0.1)
fig.show()
```

In [26]:

```
fig = px.histogram(df, x="Leak_Detection", color="Water_Pressure", title="<b>Wat
fig.update_layout(width=700, height=500, bargap=0.1)
fig.data[0].marker.color = ('#7fcdff')
fig.data[1].marker.color = ('#326ada')
fig.data[2].marker.color = ('#ff9b35')
fig.data[3].marker.color = ('#56c175')
fig.show()
```

In [27]:
```
df[df['Leak_Detection']==0].Water_Pressure.mean()
```

Out[27]:
```
298117.20404952083
```

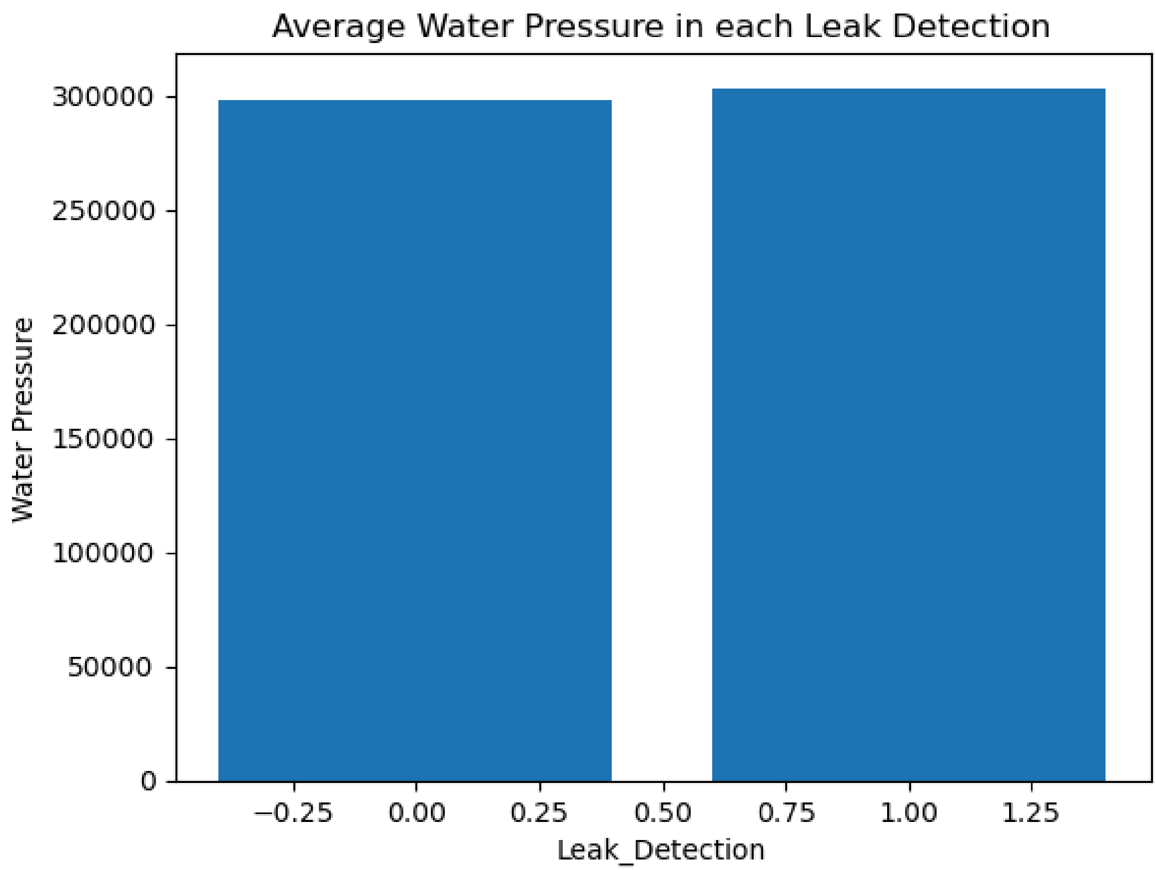
In [28]:
```
df[df['Leak_Detection']==1].Water_Pressure.mean()
```

Out[28]:
```
303498.9587578905
```

Inference: Water Pressure Rate is High if there are no leaks.

In [29]:
```
fig = px.histogram(df, x="Leak_Detection", color="Water_Level", title="<b>Water
fig.update_layout(width=700, height=500, bargap=0.1)
fig.data[0].marker.color = ('#7fcdff')
fig.data[1].marker.color = ('#326ada')
fig.data[2].marker.color = ('#ff9b35')
fig.data[3].marker.color = ('#56c175')
fig.show()
```

```
In [30]: l = [df[df['Leak_Detection']==0].Water_Pressure.mean(),df[df['Leak_Detection']==
         plt.bar(range(len(l)),l)
         plt.xlabel("Leak_Detection")
         plt.ylabel("Water Pressure")
         plt.title("Average Water Pressure in each Leak Detection")
         plt.show()
```



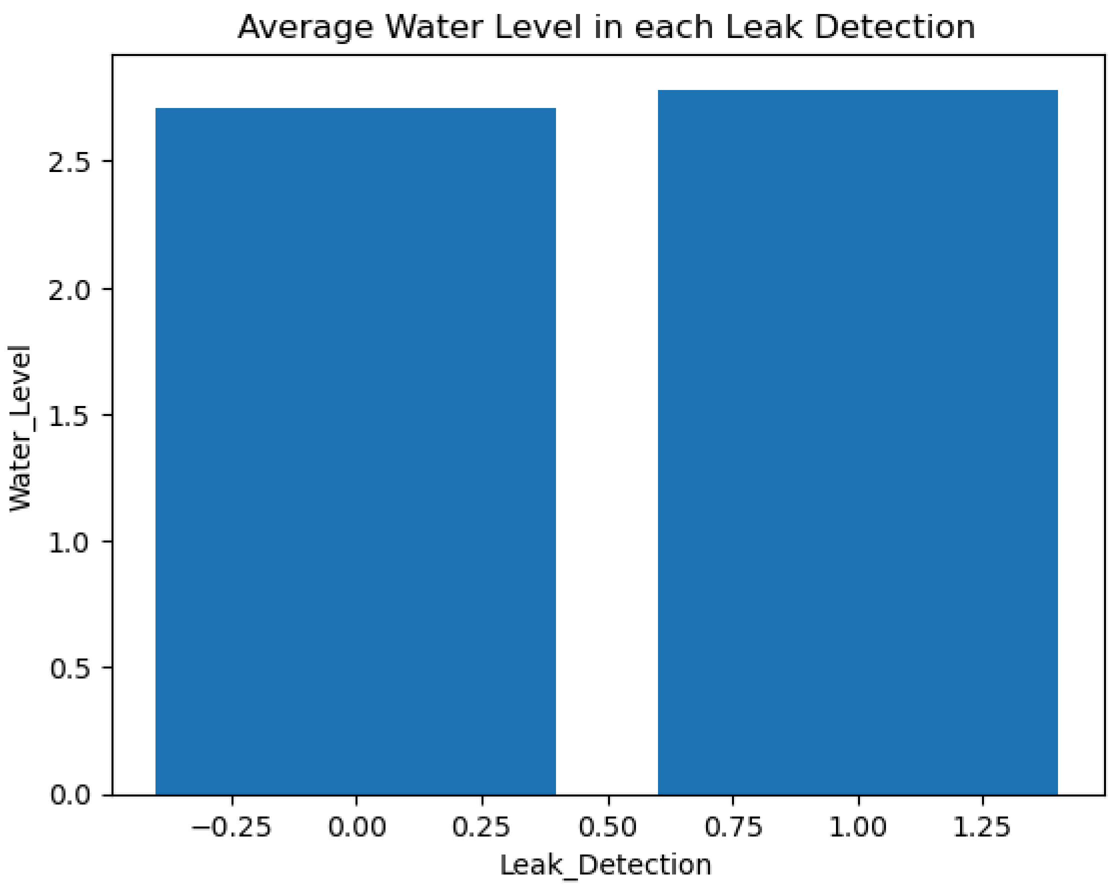
In [31]:
```
df[df['Leak_Detection']==0].Water_Level.mean()
```

Out[31]: 2.7117478292288966

In [32]:
```
df[df['Leak_Detection']==1].Water_Level.mean()
```

Out[32]: 2.7833847906642935

In [33]:
```
l = [df[df['Leak_Detection']==0].Water_Level.mean(),df[df['Leak_Detection']==1].
plt.bar(range(len(l)),l)
plt.xlabel("Leak_Detection")
plt.ylabel("Water_Level")
plt.title("Average Water Level in each Leak Detection")
plt.show()
```



Inference: Water Level is High if there are no leaks.

In [34]:
```
df['Region_ID'].value_counts()
#There are 99 Regions in the Dataset
```

Out[34]:
```
Region_ID
20    68
74    67
92    66
80    65
70    62
      ..
10    39
9     38
16    35
59    33
97    29
Name: count, Length: 99, dtype: int64
```

In [35]:
```
df['Maintenance_Status'].value_counts()
```

Out[35]:
```
Maintenance_Status
0    2554
1    2446
Name: count, dtype: int64
```

Inference: Almost 2500+ Pumps require Maintainance

In [36]:
```
df[(df['Maintenance_Status']==0) & (df['Leak_Detection']==0)]
# Use '&' instead of 'and' to combine boolean Series for element-wise comparison
# Enclose individual conditions in parentheses for clarity and proper order of o
```

Out[36]:

	Timestamp	Sensor_ID	Water_Flow_Rate	Water_Pressure	Water_Temperature
1	2024-10-18 08:05:24.179500	1740	61.706821	212911.647084	27.884570
7	2024-10-18 07:59:24.179510	1113	64.425635	102186.401817	26.757331
9	2024-10-18 07:57:24.179514	1864	82.729540	301860.149515	24.764610
18	2024-10-18 07:48:24.179526	1029	141.734346	147903.565109	28.933207
30	2024-10-18 07:36:24.179544	1790	118.683713	276827.132482	22.786617
...	...	...	...	...	...
4957	2024-10-14 21:29:24.187529	1894	182.709543	412113.031323	21.348056
4960	2024-10-14 21:26:24.187533	1920	78.304295	176430.753196	28.096811
4980	2024-10-14 21:06:24.187560	1575	114.838434	422032.921127	21.590196
4985	2024-10-14 21:01:24.187567	1679	11.892208	496888.093203	16.170737
4986	2024-10-14 21:00:24.187568	1622	14.444117	491780.722799	20.030585

1231 rows × 19 columns

In [37]:
```
df[(df['Maintenance_Status']==0) & (df['Leak_Detection']==0)].shape
```

Out[37]:
```
(1231, 19)
```

Inference: Almost 1200+ Pipes require Maintainance like Fixing the Leaks and Old Pipes usage

In [38]:
```
df[df['Pipe_Age']>50].shape
```

Out[38]:
```
(2457, 19)
```

Inference: 2000+ Pipes Age is more than 50 years. average Life Span of a pipe is 50 years.

In [39]:
```
df[df['Pump_Status']==0].shape
```

Out[39]:
```
(2480, 19)
```

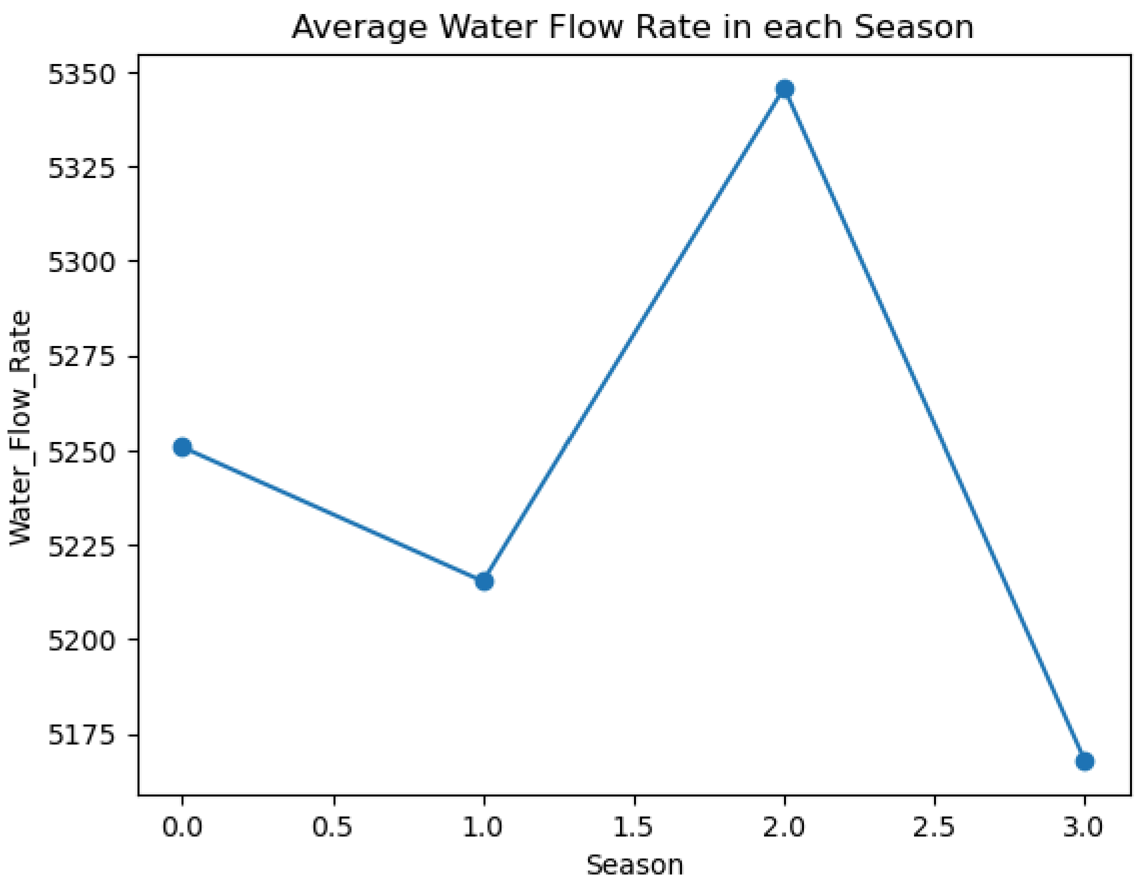
Inference: 2400+ Pumps are Inactive and They Need Maintainace

In [40]:
```
print('Average Power Consumption is', df['Power_Consumption'].mean(),'kilowats')
```

```
Average Power Consumption is 2.563528979513824 kilowats
```

In [41]:
```
fig = px.histogram(df, x="Season", color="Demand_Prediction",barmode="group" ,ti
fig.update_layout(width=700, height=500, bargap=0.1)
fig.data[0].marker.color = ('#7fcdff')
fig.data[1].marker.color = ('#326ada')
fig.data[2].marker.color = ('#ff9b35')
fig.data[3].marker.color = ('#56c175')
fig.show()
```

In [42]:
```
#Average Water Flow Rate in each season plot
l = [df[df['Season']==0].Demand_Prediction.mean(),df[df['Season']==1].Demand_Pre
plt.plot(l,marker='o')
plt.xlabel("Season")
plt.ylabel("Water_Flow_Rate")
plt.title("Average Water Flow Rate in each Season")
plt.show()
```



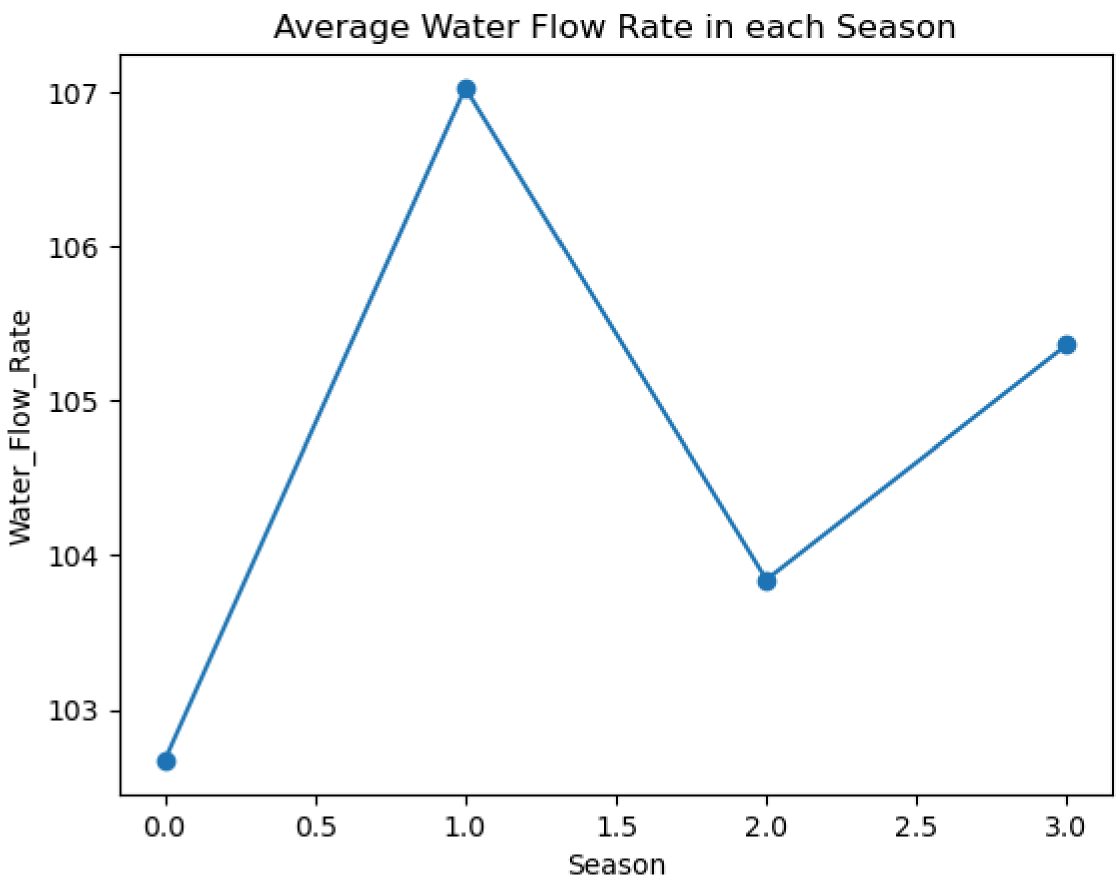
In [43]:
```
print('Demand Prediction in Fall Season is:',df[df['Season']==0].Demand_Predicti
print('Demand Prediction in Spring Season is:',df[df['Season']==1].Demand_Predic
print('Demand Prediction in Summer Season is:',df[df['Season']==2].Demand_Predic
print('Demand Prediction in Winter Season is:',df[df['Season']==3].Demand_Predic
```

```
Demand Prediction in Fall Season is: 5250.820063348908
Demand Prediction in Spring Season is: 5215.353987850937
Demand Prediction in Summer Season is: 5345.736913642295
Demand Prediction in Winter Season is: 5167.9806321677925
```

Inference: The Demand for Water is More in Summer

In [44]:
```
fig = px.histogram(df, x="Season", color="Water_Flow_Rate",barmode="group" ,titl
fig.update_layout(width=700, height=500, bargap=0.1)
# fig.data[0].marker.color = ('#7fcdff')
# fig.data[1].marker.color = ('#326ada')
# fig.data[2].marker.color = ('#ff9b35')
# fig.data[3].marker.color = ('#56c175')
fig.show()
```

In [45]:
```
#Average Water Flow Rate in each season plot
l = [df[df['Season']==0].Water_Flow_Rate.mean(),df[df['Season']==1].Water_Flow_R
plt.plot(l,marker='o')
plt.xlabel("Season")
plt.ylabel("Water_Flow_Rate")
plt.title("Average Water Flow Rate in each Season")
plt.show()
```



Inference: The Demand for Water is High in Summer

In [46]:
```
print('Demand Prediction in Fall Season is:',df[df['Season']==0].Water_Flow_Rate
print('Demand Prediction in Spring Season is:',df[df['Season']==1].Water_Flow_Ra
print('Demand Prediction in Summer Season is:',df[df['Season']==2].Water_Flow_Ra
print('Demand Prediction in Winter Season is:',df[df['Season']==3].Water_Flow_Ra
```

```
Demand Prediction in Fall Season is: 102.67176537387698
Demand Prediction in Spring Season is: 107.02558048308573
Demand Prediction in Summer Season is: 103.84233065489778
Demand Prediction in Winter Season is: 105.36204703642579
```

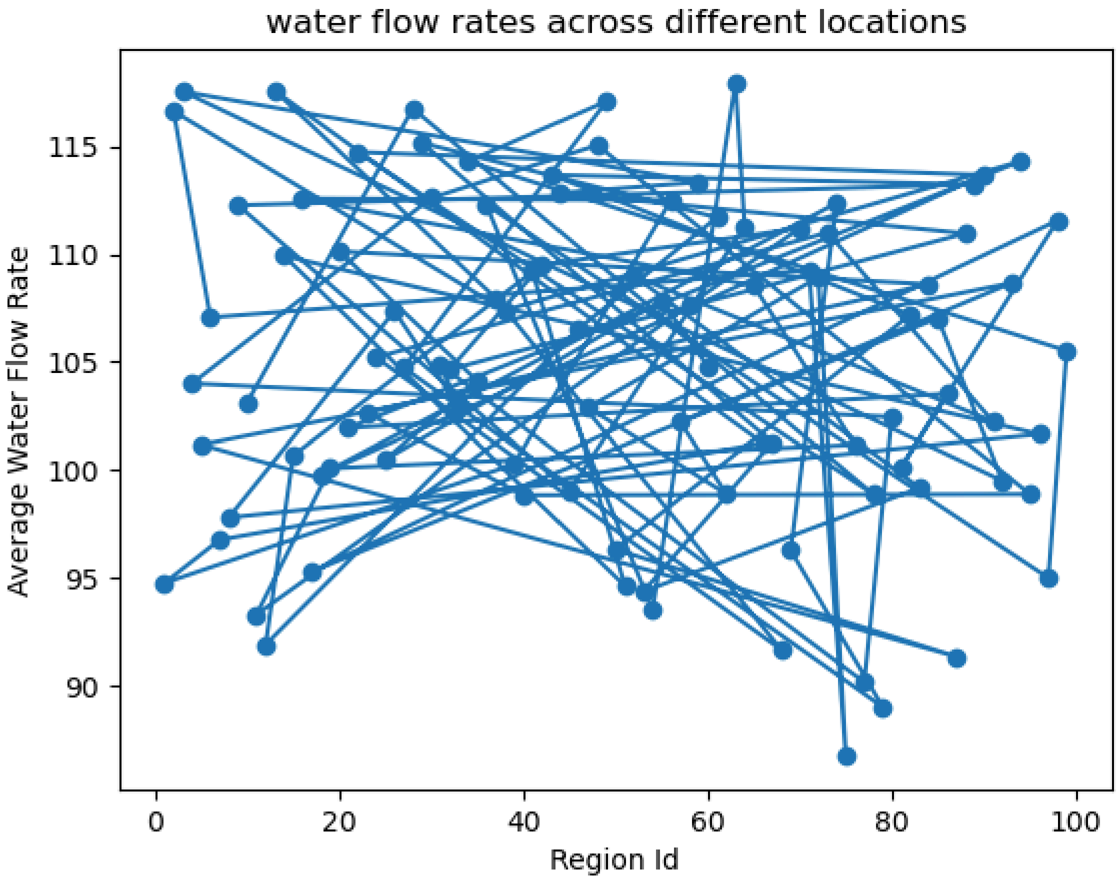
Inference: The Water Flow Rate is less in Summer. So, Water Scarcity takes place in Summer.

In [47]:
```
fig = px.histogram(df, x="Season", color="Water_Pressure",barmode="group" ,title
fig.update_layout(width=700, height=500, bargap=0.1)
fig.data[0].marker.color = ('#7fcdff')
fig.data[1].marker.color = ('#326ada')
fig.data[2].marker.color = ('#ff9b35')
fig.data[3].marker.color = ('#56c175')
fig.show()
```

In [63]:

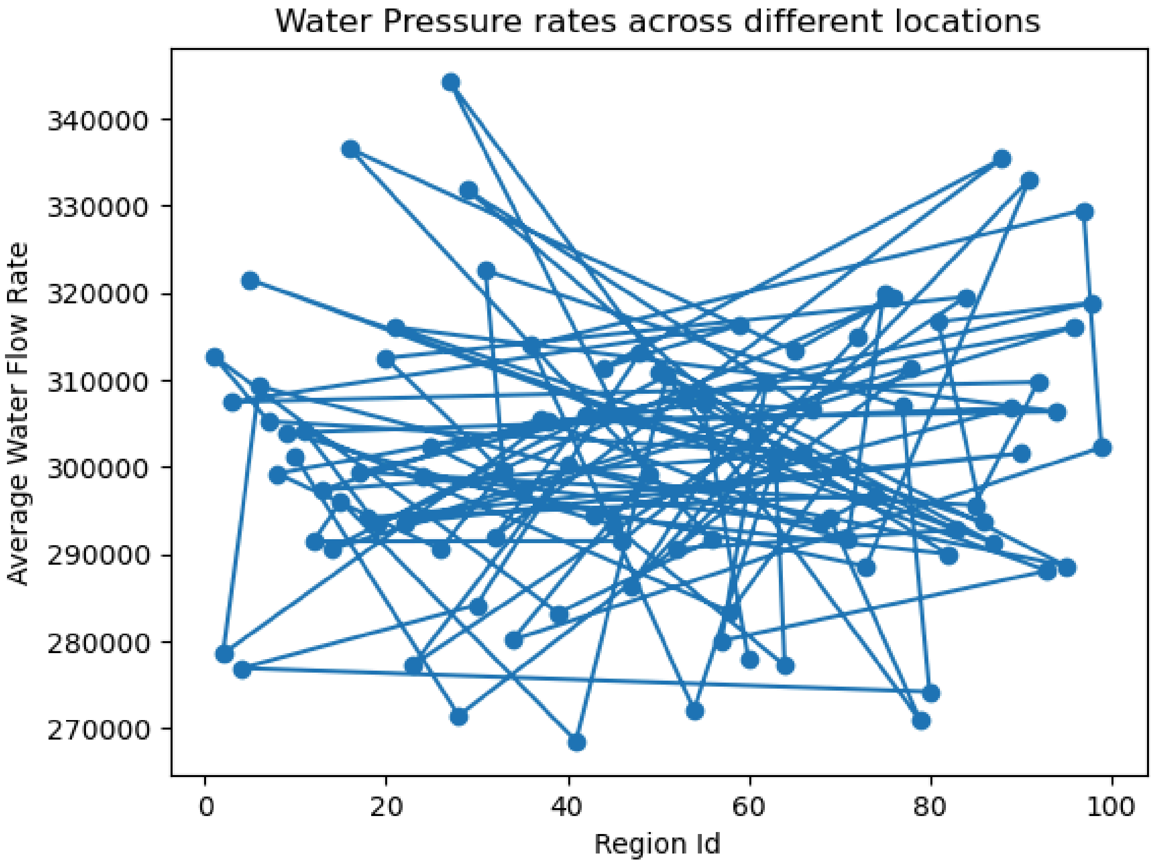
```
# water flow rates across different locations.

l = []
t = list(df['Region_ID'].unique())
avg = [df[df['Region_ID']==i].Water_Flow_Rate.mean() for i in t]
plt.plot(t,avg,marker='o')
plt.xlabel('Region Id')
plt.ylabel('Average Water Flow Rate')
plt.title('water flow rates across different locations')
plt.show()
```



In [65]:
```
# water pressure rates across different locations.

l = []
t = list(df['Region_ID'].unique())
avg = [df[df['Region_ID']==i].Water_Pressure.mean() for i in t]
plt.plot(t,avg,marker='o')
plt.xlabel('Region Id')
plt.ylabel('Average Water Flow Rate')
plt.title('Water Pressure rates across different locations')
plt.show()
```



Inferences

1. 994 Sensors are used to collect the data
2. Average Water Flow Rate is 104 liters
3. Maximum and Minimum WaterFlow Rates are : 199.94073540452425, 10.05200204771487 liters/minute 5<>/li
4. Average Water Pressure: 300842.52463383926 Pascals
5. Minimum Water Pressure: 100050.2226882317 Pascals
6. Maximum Water Pressure: 499933.19281597 Pascals76
7. Demand for Water Quality 5 is High
8. 2000+ Pipes are Having Water Leakage Problem
9. Water Flow Rate is High if Leaks are not there when compared to Water Flow rate if leaks exist
10. Water Pressure Rate is High if there are no leaks.
11. Water Level is High if there are no leaks.
12. Almost 2500+ Pumps require Maintainance
13. Almost 1200+ Pipes require Maintainance like Fixing the Leaks and Old Pipes usage
14. 2000+ Pipes Age is more than 50 years. average Life Span of a pipe is 50 years.Usage of Old Pipes in Most of the Regions
15. 2400+ Pumps are Inactive and They Need Maintainace
16. The Demand for Water is High in Summer
17. The Water Flow Rate is less in Summer. So, Water Scarcity exisn Summer. i>>

>

Anomaly Detection using Isolation Forest

Anomaly Detection refers to identifying data points or patterns that deviate significantly from the expected behavior of a system or dataset. These anomalies (or outliers) can indicate unusual events, faults, or rare occurrences. In industrial processes, anomaly detection is critical for identifying system malfunctions, operational inefficiencies, or potential threats before they cause significant issues.

In [53]:
```
from sklearn.ensemble import IsolationForest

# Features for anomaly detection
features = ['Water_Flow_Rate', 'Water_Pressure', 'Water_Temperature', 'Water_Qua
X = df[features]

# Fit Isolation Forest
iso_forest = IsolationForest(contamination=0.05, random_state=42)
df['Anomaly'] = iso_forest.fit_predict(X)

# Show anomalies
df = df[df['Anomaly'] != -1]
```

In [54]:
```
df
```

Out[54]:

	Timestamp	Sensor_ID	Water_Flow_Rate	Water_Pressure	Water_Temperature
0	2024-10-18 08:06:24.179489	1936	123.012581	458742.877621	11.436859
1	2024-10-18 08:05:24.179500	1740	61.706821	212911.647084	27.884570
2	2024-10-18 08:04:24.179502	1416	59.250224	163731.693406	21.065049
3	2024-10-18 08:03:24.179503	1441	81.023858	457735.726387	22.790899
4	2024-10-18 08:02:24.179505	1163	67.688319	320732.978098	14.563713
...	...	...	...	...	...
4993	2024-10-14 20:53:24.187581	1759	168.756434	442702.318549	10.971687
4995	2024-10-14 20:51:24.187583	1294	180.910228	103536.061304	16.736808
4997	2024-10-14 20:49:24.187586	1322	150.795986	406852.689983	15.348981
4998	2024-10-14 20:48:24.187587	1744	133.151782	325735.766144	11.608285
4999	2024-10-14 20:47:24.187589	1359	198.443786	344624.930448	22.842345

4512 rows × 20 columns

In [55]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
```

In [56]:

```
features = ['Sensor_ID', 'Water_Flow_Rate', 'Water_Pressure', 'Water_Temperature
X = df[features]
y = df['Demand_Prediction']
# Train the model
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X, y)
```

Out[56]:

```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

Forecasting the Water Demand

In [58]:

```
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt

# Select water demand (daily usage) as the time series to forecast
```

```
df['Timestamp'] = pd.to_datetime(df['Timestamp'])
df.set_index('Timestamp', inplace=True)
daily_usage = df['Daily_Usage'].resample('D').sum()

# Fit ARIMA model
model = ARIMA(daily_usage, order=(5,1,0))
arima_result = model.fit()

# Forecast the next 30 days
forecast = arima_result.forecast(steps=30)

# Plot the forecast
plt.figure(figsize=(10, 6))
plt.plot(daily_usage, label='Observed')
plt.plot(forecast, label='Forecast', linestyle='--')
plt.title('Water Demand Prediction')
plt.legend()
plt.show()
```

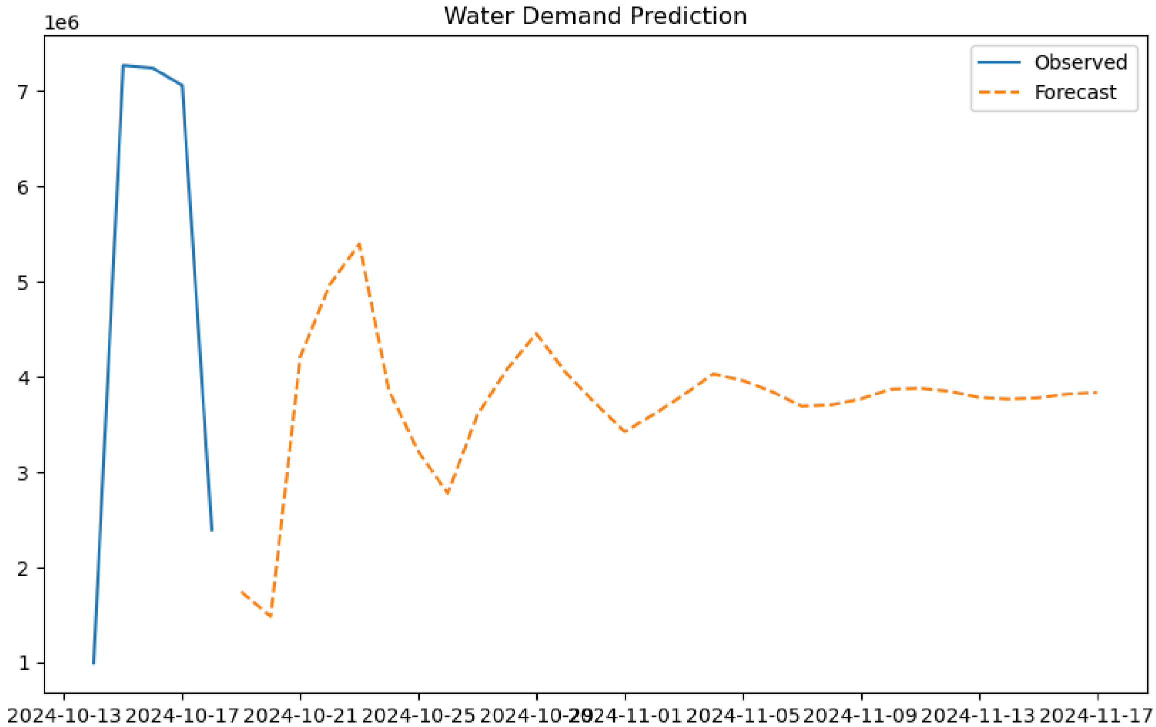
```
C:\Users\VENKATA ARJUN\AppData\Local\Temp\ipykernel_10524\3275193480.py:5: Settin
gWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stabl
e/user_guide/indexing.html#returning-a-view-versus-a-copy

C:\Users\VENKATA ARJUN\anaconda3\Lib\site-packages\statsmodels\tsa\statespace\sar
imax.py:866: UserWarning:

Too few observations to estimate starting parameters for ARMA and trend. All para
meters except for variances will be set to zeros.
```



In [ ]:

```
df.head()
```

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

In [ ]:
```
```