In [1]:

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

In [2]:

```
df = pd.read_csv('Data/Surface_Water_Quality_Data_1995_through_December_2022.csv', encod
```

```
C:\Users\ravir\AppData\Local\Temp\ipykernel_26104\3601453312.py:1: DtypeWa
rning: Columns (3) have mixed types. Specify dtype option on import or set
low_memory=False.
  df = pd.read_csv('Data/Surface_Water_Quality_Data_1995_through_December_
2022.csv', encoding='ISO-8859-1')
```

In [3]:

```
#To display the first 5 rows of the DataFrame
df.head()
```

Out[3]:

	OBJECTID	Station	GPS Coordinate North	GPS Coordinate West	Parameter	Lab	Result	Unit
0	1	HAMILTON AVE.	39.33673	-76.53967	Copper_Total	Martel	10	ug/L
1	2	HAMILTON AVE.	39.33673	-76.53967	Fecal Coliform	Martel	2400	MPN/100ml
2	3	HAMILTON AVE.	39.33673	-76.53967	Lead_Total	Martel	40	ug/L
3	4	HAMILTON AVE.	39.33673	-76.53967	Oil & Grease	Martel	2.8	mg/L
4	5	HAMILTON AVE.	39.33673	-76.53967	Zinc_Total	Martel	20	ug/L

In [4]:

```
#To display the last 5 rows of the DataFrame
df.tail()
```

Out[4]:

	OBJECTID	Station	GPS Coordinate North	GPS Coordinate West	Parameter	Lab	Result	Unit	
428101	428102	STONY RUN @ GILMAN	39.36086	-76.62985	Conductivity	WQM Field	572	umhos	12
428102	428103	STONY RUN @ GILMAN	39.36086	-76.62985	Dissolved Oxygen	WQM Field	11.8	mg/L	12
428103	428104	STONY RUN @ GILMAN	39.36086	-76.62985	Hach Ammonia-Nitrogen	WQM Field	0	mg/L	12
428104	428105	STONY RUN @ GILMAN	39.36086	-76.62985	pH	WQM Field	7.42	pH units	12
428105	428106	STONY RUN @ GILMAN	39.36086	-76.62985	Water Temperature	WQM Field	8.9	degrees Celsius	12

In [5]:

```
#To display the total number of missing values in each column in DataFrame.
df.isnull().sum()
```

Out[5]:

```
OBJECTID                0
Station                 0
GPS Coordinate North    0
GPS Coordinate West     0
Parameter               0
Lab                     0
Result                  0
Unit                    0
datetime                0
dtype: int64
```

In [6]:

```
#  To display a statistical summary of the numerical columns in a pandas DataFrame.
df.describe()
```

Out[6]:

	OBJECTID	GPS Coordinate North
count	428106.000000	428106.000000
mean	214053.500000	39.314712
std	123583.701507	0.033675
min	1.000000	39.239460
25%	107027.250000	39.283190
50%	214053.500000	39.314860
75%	321079.750000	39.339430
max	428106.000000	39.371730

In [7]:

```
# To display returns the data type of each column in the DataFrame.
df.dtypes
```

Out[7]:

```
OBJECTID                  int64
Station                  object
GPS Coordinate North    float64
GPS Coordinate West      object
Parameter                object
Lab                      object
Result                   object
Unit                     object
datetime                 object
dtype: object
```

In [8]:

```
# To provides a summary of the DataFrame's structure
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 428106 entries, 0 to 428105
Data columns (total 9 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   OBJECTID              428106 non-null  int64  
 1   Station               428106 non-null  object 
 2   GPS Coordinate North  428106 non-null  float64
 3   GPS Coordinate West   428106 non-null  object 
 4   Parameter             428106 non-null  object 
 5   Lab                   428106 non-null  object 
 6   Result                428106 non-null  object 
 7   Unit                  428106 non-null  object 
 8   datetime              428106 non-null  object 
dtypes: float64(1), int64(1), object(7)
memory usage: 29.4+ MB
```

In [9]:

```
df['OBJECTID'] = df['OBJECTID'].astype('int32')
df['GPS Coordinate North'] = df['GPS Coordinate North'].astype('float32')
df['GPS Coordinate West'] = df['GPS Coordinate West'].astype('float32')
df['datetime'] = pd.to_datetime(df['datetime'])
df['Unit'] = df['Unit'].astype('category')
df['Parameter'] = df['Parameter'].astype('category')
df['Lab'] = df['Lab'].astype('category')
df['Station'] = df['Station'].astype('category')
df['Result'] = df['Result'].astype('category')
```

The code changes the DataFrame's selected columns' data types to those that are suitable for analysis. An illustration would be changing a column of floats to the more memory-effective data type float32.

In [10]:

```
# Re print the datatype
print(df.dtypes)
```

```
OBJECTID                         int32
Station                       category
GPS Coordinate North           float32
GPS Coordinate West            float32
Parameter                     category
Lab                           category
Result                        category
Unit                          category
datetime                datetime64[ns]
dtype: object
```

In [11]:

```
#This code returns a list of the column names in the DataFrame
df.columns
```

Out[11]:

```
Index(['OBJECTID', 'Station', 'GPS Coordinate North', 'GPS Coordinate Wes
t',
       'Parameter', 'Lab', 'Result', 'Unit', 'datetime'],
      dtype='object')
```

In [12]:

```
# convert datetime column to datetime data type
df['datetime'] = pd.to_datetime(df['datetime'])

# create new columns for month, day, year, hour, and minute
df['month'] = df['datetime'].dt.month
df['day'] = df['datetime'].dt.day
df['year'] = df['datetime'].dt.year
df['hour'] = df['datetime'].dt.hour
df['minute'] = df['datetime'].dt.minute

# drop original datetime column
df = df.drop('datetime', axis=1)
# show result
print(df.head())
```

```
   OBJECTID       Station  GPS Coordinate North  GPS Coordinate West  \
0         1  HAMILTON AVE.             39.336731           -76.539673
1         2  HAMILTON AVE.             39.336731           -76.539673
2         3  HAMILTON AVE.             39.336731           -76.539673
3         4  HAMILTON AVE.             39.336731           -76.539673
4         5  HAMILTON AVE.             39.336731           -76.539673

        Parameter     Lab Result       Unit  month  day  year  hour  minut
e
0    Copper_Total  Martel     10       ug/L      4    3  1995    13      3
0
1  Fecal Coliform  Martel   2400  MPN/100ml      4    3  1995    13      3
0
2      Lead_Total  Martel     40       ug/L      4    3  1995    13      3
0
3    Oil & Grease  Martel    2.8       mg/L      4    3  1995    13      3
0
4      Zinc_Total  Martel     20       ug/L      4    3  1995    13      3
0
```

The month, day, year, hour, and minute are then extracted from the datetime column and added to new columns using this code, which first transforms a datetime column to a datetime data type. The DataFrame's old datetime column is then deleted, and the first few rows of the new DataFrame are printed.

In [13]:

```
df.head()
```

Out[13]:

	OBJECTID	Station	GPS Coordinate North	GPS Coordinate West	Parameter	Lab	Result	Unit	r
0	1	HAMILTON AVE.	39.336731	-76.539673	Copper_Total	Martel	10	ug/L	
1	2	HAMILTON AVE.	39.336731	-76.539673	Fecal Coliform	Martel	2400	MPN/100ml	
2	3	HAMILTON AVE.	39.336731	-76.539673	Lead_Total	Martel	40	ug/L	
3	4	HAMILTON AVE.	39.336731	-76.539673	Oil & Grease	Martel	2.8	mg/L	
4	5	HAMILTON AVE.	39.336731	-76.539673	Zinc_Total	Martel	20	ug/L	

In [14]:

```
#The resulting DataFrame contains only those columns.
df[['Station', 'Parameter', 'Result']]
```

Out[14]:

	Station	Parameter	Result
0	HAMILTON AVE.	Copper_Total	10
1	HAMILTON AVE.	Fecal Coliform	2400
2	HAMILTON AVE.	Lead_Total	40
3	HAMILTON AVE.	Oil & Grease	2.8
4	HAMILTON AVE.	Zinc_Total	20
...	...	...	...
428101	STONY RUN @ GILMAN	Conductivity	572
428102	STONY RUN @ GILMAN	Dissolved Oxygen	11.8
428103	STONY RUN @ GILMAN	Hach Ammonia-Nitrogen	0
428104	STONY RUN @ GILMAN	pH	7.42
428105	STONY RUN @ GILMAN	Water Temperature	8.9

428106 rows × 3 columns

In [15]:

```
df.groupby('Parameter').mean()
df.groupby(['Station', 'Parameter']).max()
```

```
C:\Users\ravir\AppData\Local\Temp\ipykernel_26104\788452635.py:2: FutureWa
rning: Dropping invalid columns in DataFrameGroupBy.max is deprecated. In
a future version, a TypeError will be raised. Before calling .max, select
only columns which should be valid for the function.
  df.groupby(['Station', 'Parameter']).max()
```

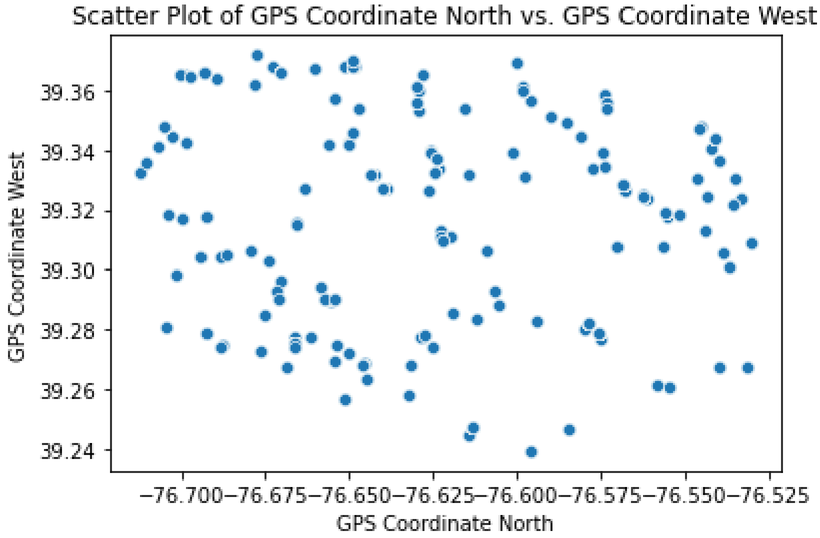
Out[15]:

Station	Parameter	OBJECTID	GPS Coordinate North	GPS Coordinate West	month	day	year
1201 S PACA ST	**Ammonia-Nitrogen**	NaN	NaN	NaN	NaN	NaN	NaN
	Antimony_Dissolved	NaN	NaN	NaN	NaN	NaN	NaN
	Antimony_Total	NaN	NaN	NaN	NaN	NaN	NaN
	Arsenic_Dissolved	NaN	NaN	NaN	NaN	NaN	NaN
	Arsenic_Total	NaN	NaN	NaN	NaN	NaN	NaN
...	**...**	...	...	...	...	...	...
WYNDHURST AVE.	**Zinc, Total**	NaN	NaN	NaN	NaN	NaN	NaN
	Zinc_Dissolved	NaN	NaN	NaN	NaN	NaN	NaN
	Zinc_Total	NaN	NaN	NaN	NaN	NaN	NaN
	chlorine	NaN	NaN	NaN	NaN	NaN	NaN
	pH	426274.0	39.353279	-76.629562	12.0	31.0	2022.0

7938 rows × 8 columns

In [16]:

```
## Creating a scatter plot of 'GPS Coordinate North' vs. 'GPS Coordinate West'
sns.scatterplot(x='GPS Coordinate West', y='GPS Coordinate North', data=df)
plt.title('Scatter Plot of GPS Coordinate North vs. GPS Coordinate West')
plt.xlabel('GPS Coordinate North')
plt.ylabel('GPS Coordinate West')
plt.show()
```



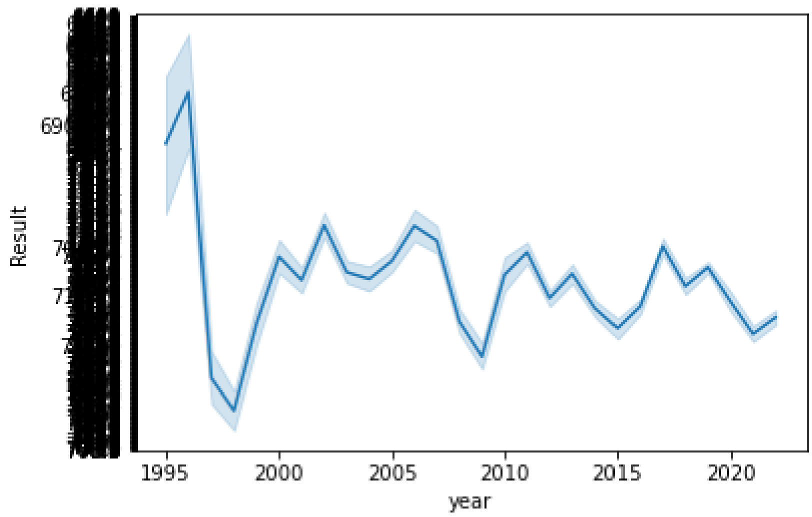
This code creates a scatter plot of GPS Coordinate North vs GPS Coordinate West using Seaborn library and Matplotlib. The plot can help visualize any patterns or relationships between the two variables.

In [17]:

```
#creates a line plot of the 'Result' column against the 'year' column
sns.lineplot(x='year', y='Result', data=df[df['Parameter'] == 'pH'])
```

Out[17]:

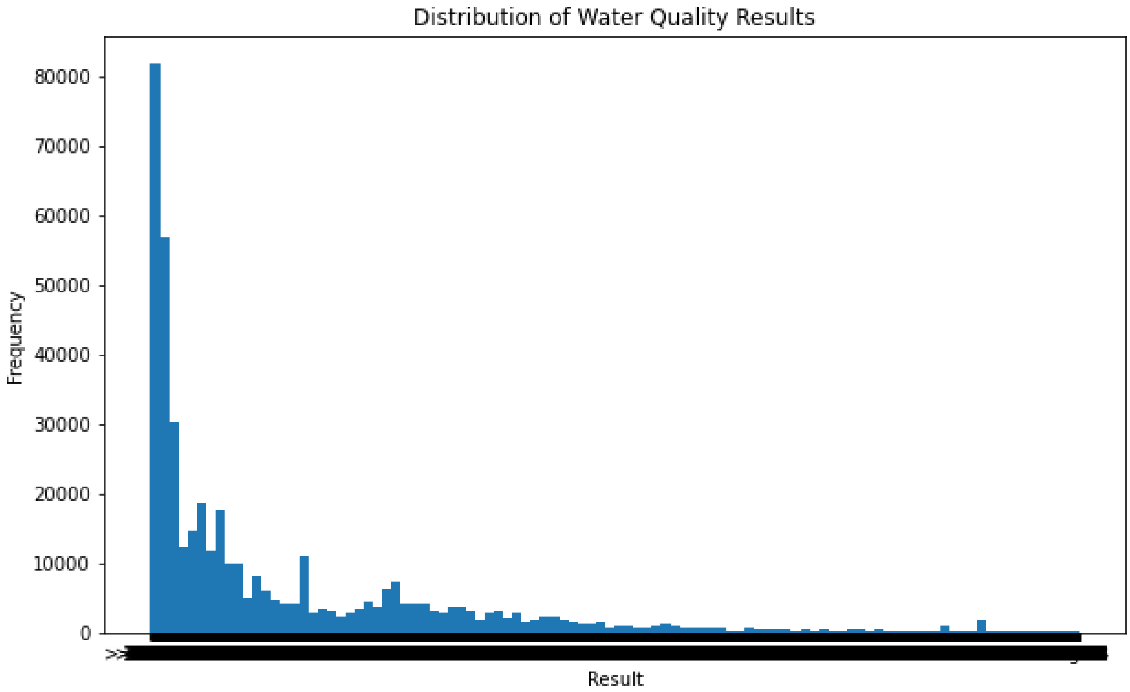
```
<AxesSubplot:xlabel='year', ylabel='Result'>
```



The help of the Seaborn library, this code generates a line plot where the y-axis displays the pH parameter result value and the x-axis the year. Before graphing, the data is filtered to include only rows having the pH parameter. The line plot depicts the pH value's evolution over time.

In [18]:

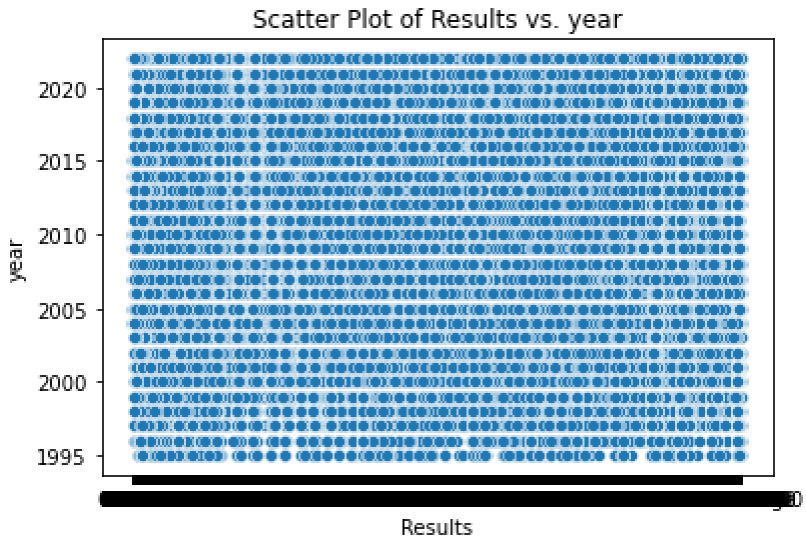
```
# Create a histogram of the 'Result' column
plt.figure(figsize=(10, 6))
plt.hist(df.Result,bins=100)
plt.title('Distribution of Water Quality Results')
plt.xlabel('Result')
plt.ylabel('Frequency')
plt.show()
```



This code creates a histogram of the DataFrame's 'Result' column with 100 bins. The distribution of water quality results and the frequency of occurrence for each result are represented visually.

In [19]:

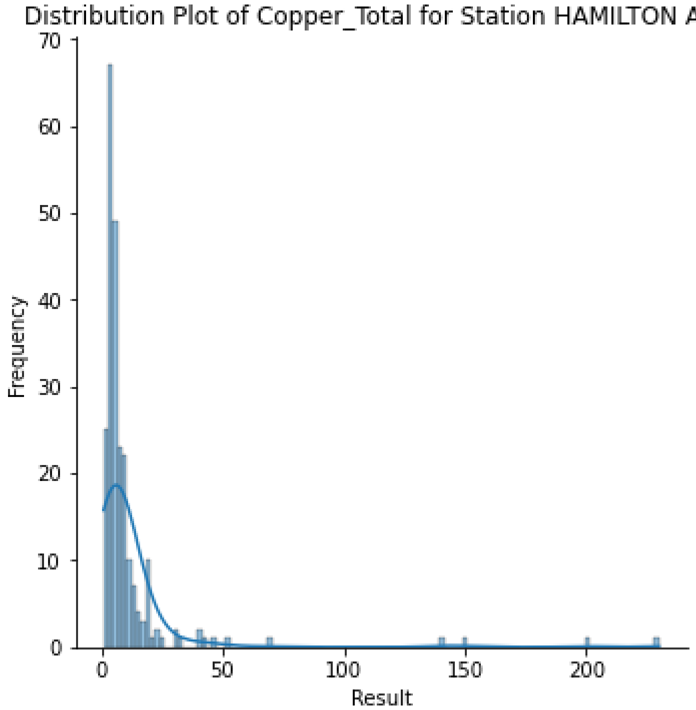
```
#Creating a scatter plot of 'Results' vs. 'year'
sns.scatterplot(x='Result', y='year', data=df)
plt.title('Scatter Plot of Results vs. year')
plt.xlabel('Results')
plt.ylabel('year')
plt.show()
```



The DataFrame 'df', 'Results' column is plotted against the 'year' column using this code. The plot title, x-axis label, and y-axis label are all set using the seaborn library.

In [20]:

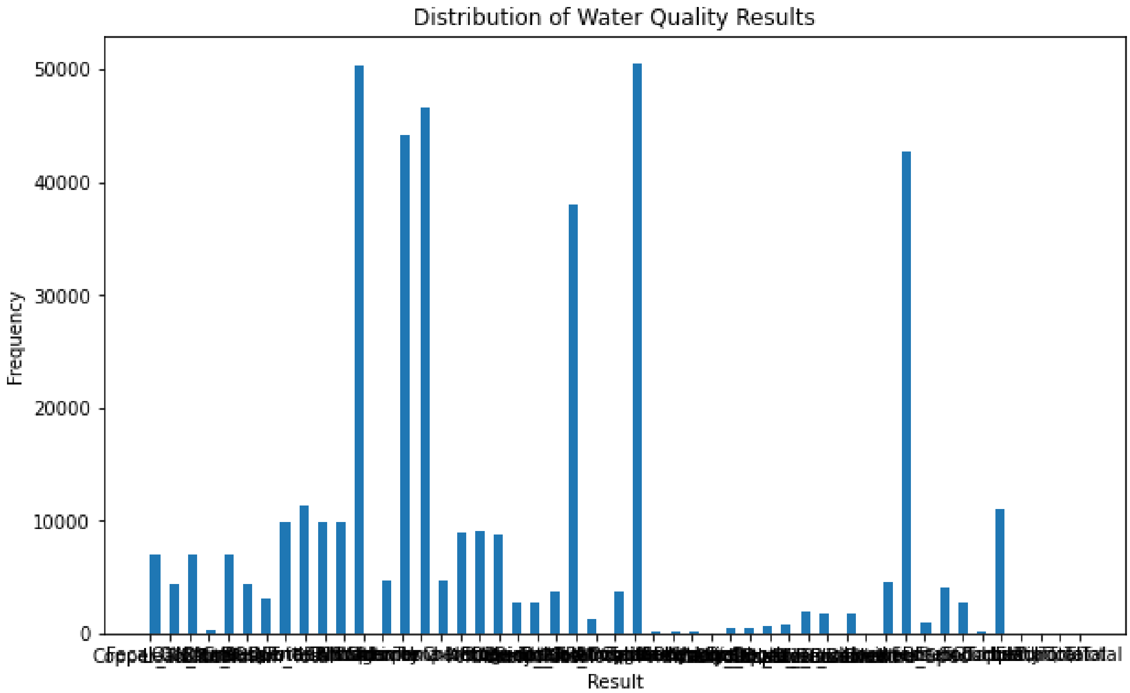
```
# Create a line plot of 'datetime' vs. 'Result' for a specific station and parameter
station = 'HAMILTON AVE.'
parameter = 'Copper_Total'
df_filtered = df[(df['Station'] == station) & (df['Parameter'] == parameter)]
df_filtered = df_filtered.dropna(subset=['Result'])
df_filtered['Result'] = pd.to_numeric(df_filtered['Result'], errors='coerce')
sns.displot(data=df_filtered, x='Result', kde=True)
plt.title(f'Distribution Plot of {parameter} for Station {station}')
plt.xlabel('Result')
plt.ylabel('Frequency')
plt.show()
```



The 'Copper_Total' parameter for the 'HAMILTON AVE.' station is shown as a distribution using the code below. The 'Result' column is then converted to a numeric data type after the DataFrame is first filtered to only contain data for that particular station and parameter. Any rows that have missing values in the 'Result' column are then removed. The frequency distribution of the 'Result' values for that particular station and parameter is displayed on the resultant distribution plot.

In [21]:

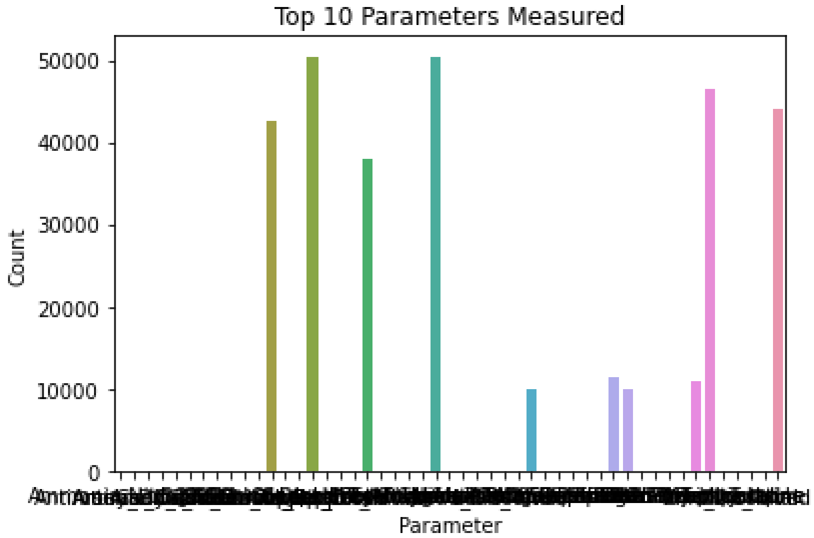
```
# Create a histogram of the 'Parameter' column
plt.figure(figsize=(10, 6))
plt.hist(df.Parameter,bins=100)
plt.title('Distribution of Water Quality Results')
plt.xlabel('Result')
plt.ylabel('Frequency')
plt.show()
```



This code generates a histogram that displays the values in the DataFrame 'df''s 'Parameter' column's frequency distribution. The plot displays the quantity of each distinct value in the column.

In [22]:

```
# Create a bar plot of the top 10 parameters measured in the dataset
top_parameters = df['Parameter'].value_counts().nlargest(10)
sns.barplot(x=top_parameters.index, y=top_parameters.values)
plt.title('Top 10 Parameters Measured')
plt.xlabel('Parameter')
plt.ylabel('Count')
plt.show()
```



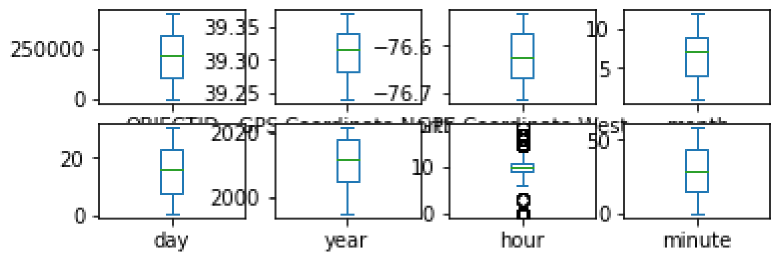
Based on the count of each parameter, this code generates a bar plot that displays the top 10 parameters measured in the dataset. The parameter names are shown on the x-axis, while the parameter counts are shown on the y-axis.

In [23]:

```
# Plot the box plots of the features
df.plot(kind='box', subplots=True, layout=(4,4), sharex=False, sharey=False)
```

Out[23]:

```
OBJECTID                  AxesSubplot(0.125,0.71587;0.168478x0.16413)
GPS Coordinate North   AxesSubplot(0.327174,0.71587;0.168478x0.16413)
GPS Coordinate West    AxesSubplot(0.529348,0.71587;0.168478x0.16413)
month                  AxesSubplot(0.731522,0.71587;0.168478x0.16413)
day                      AxesSubplot(0.125,0.518913;0.168478x0.16413)
year                  AxesSubplot(0.327174,0.518913;0.168478x0.16413)
hour                  AxesSubplot(0.529348,0.518913;0.168478x0.16413)
minute                AxesSubplot(0.731522,0.518913;0.168478x0.16413)
dtype: object
```



Box plots are produced by this code for each feature (column) in the dataset. Each plot has the same x and y axes and is structured in a 4 by 4 grid. Box plots are used to show the distribution of values for each characteristic and spot any outliers.

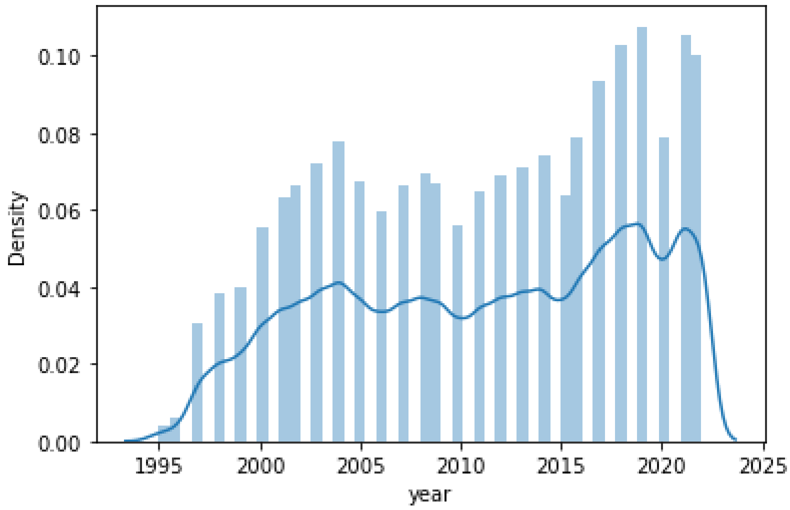
In [24]:

```
sns.distplot(df['year'])
```

```
D:\anaconda\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future ver
sion. Please adapt your code to use either `displot` (a figure-level funct
ion with similar flexibility) or `histplot` (an axes-level function for hi
stograms).
  warnings.warn(msg, FutureWarning)
```

Out[24]:

```
<AxesSubplot:xlabel='year', ylabel='Density'>
```



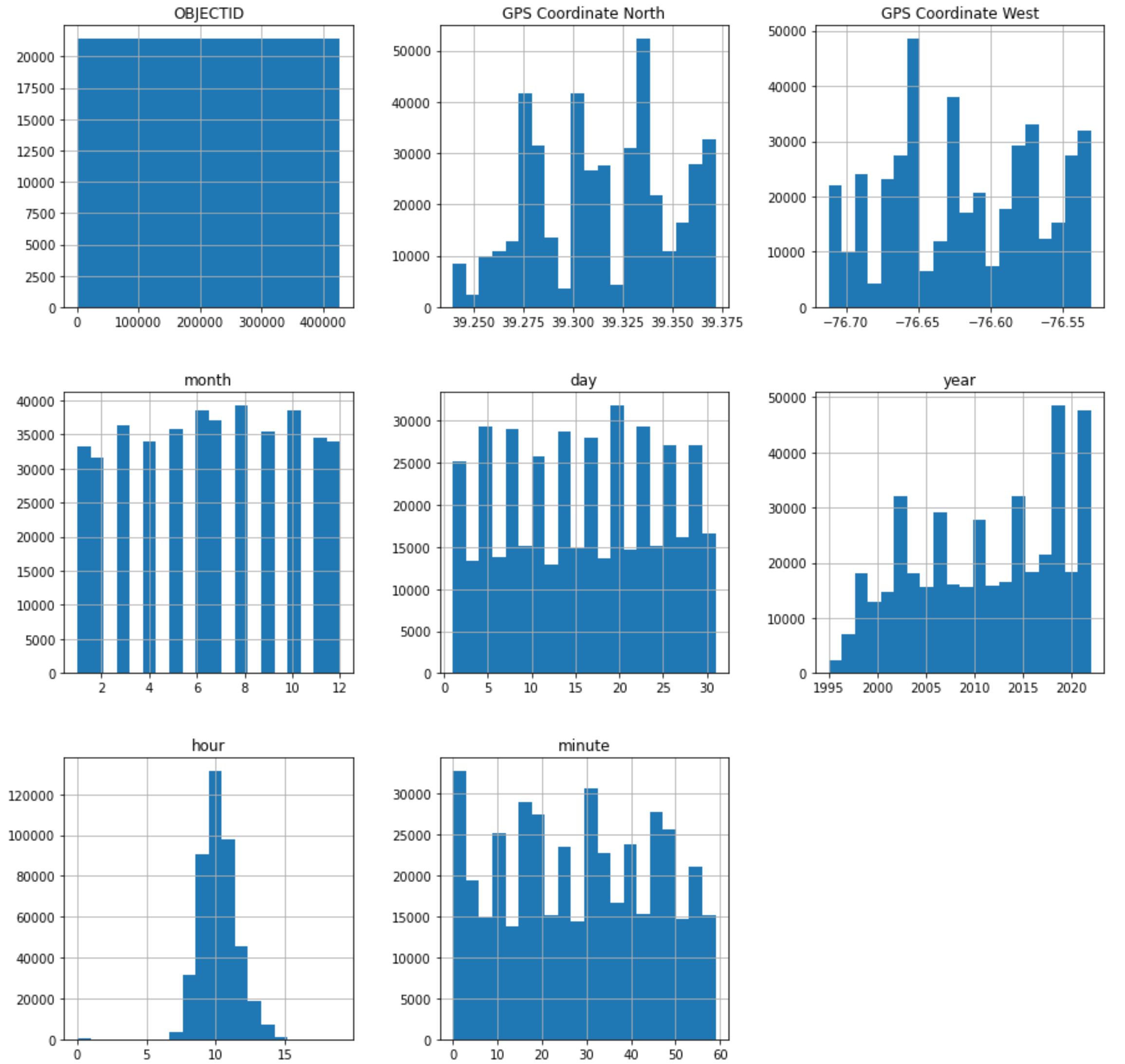
The 'year' column in the DataFrame 'df' is given a distribution plot using this code, which displays the distribution of the years in which the water quality measurements were made.

In [25]:

```
# Plot the histograms of the features
df.hist(bins=20, figsize=(15,15))
```

Out[25]:

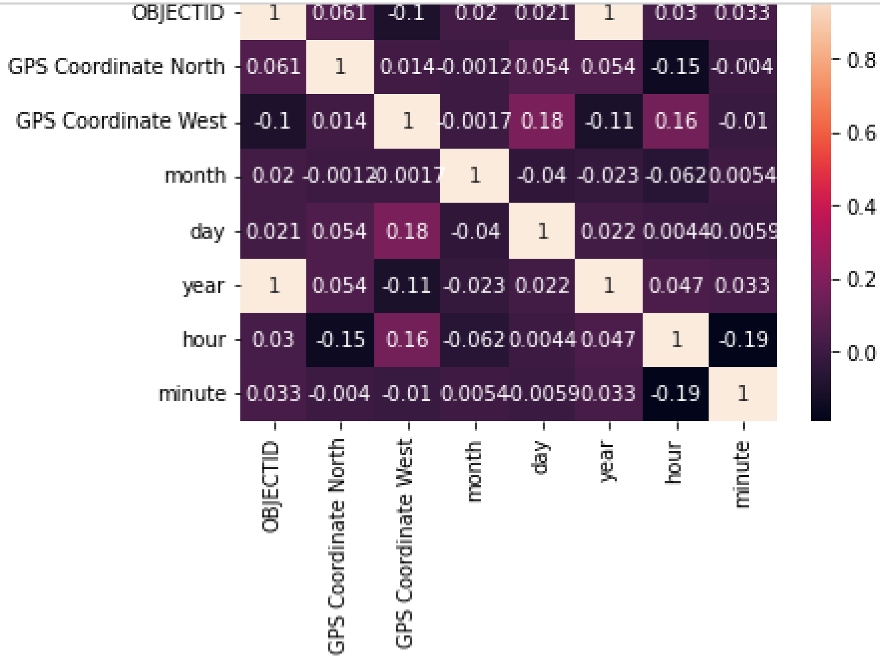
```
array([[<AxesSubplot:title={'center':'OBJECTID'}>,
        <AxesSubplot:title={'center':'GPS Coordinate North'}>,
        <AxesSubplot:title={'center':'GPS Coordinate West'}>],
       [<AxesSubplot:title={'center':'month'}>,
        <AxesSubplot:title={'center':'day'}>,
        <AxesSubplot:title={'center':'year'}>],
       [<AxesSubplot:title={'center':'hour'}>,
        <AxesSubplot:title={'center':'minute'}>, <AxesSubplot:>]],
      dtype=object)
```



With the x-axis indicating the range of values and the y-axis representing the frequency of those values, this code generates a grid of histograms for each numerical column in the dataset. The number of bins or bars used in the histogram is controlled by the "bins" option. The plot's size is determined by the "figsize" option.

In [26]:

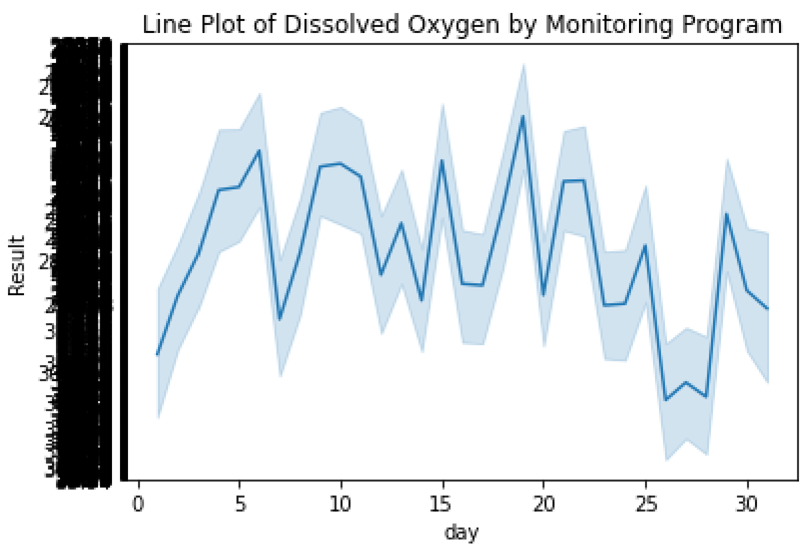
```
# Create a heatmap of the correlation matrix for all numeric columns in the dataset
corr_matrix = df.select_dtypes(include='number').corr()
sns.heatmap(corr_matrix, annot=True)
plt.title('Correlation Matrix')
plt.show()
```



The correlation matrix of each numerical column in the dataset is shown using a heatmap created by this code. The actual correlation coefficient is shown in each cell's annotation, whereas the color of each cell denotes how strongly two variables are correlated.

In [27]:

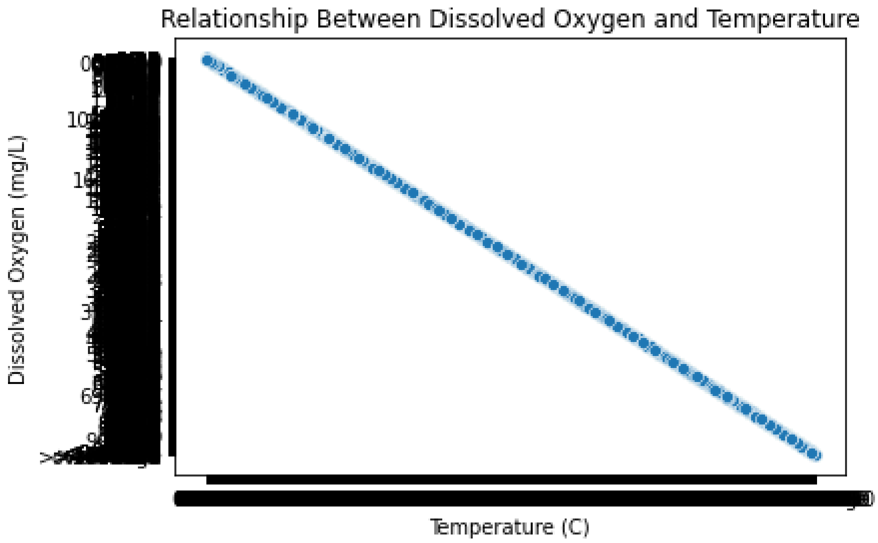
```
# Create a line plot of 'day' vs. 'Result' for a specific parameter, grouped by the moni
parameter = 'Dissolved Oxygen'
df_filtered = df[df['Parameter'] == parameter]
sns.lineplot(x='day', y='Result', data=df_filtered)
plt.title(f'Line Plot of {parameter} by Monitoring Program')
plt.xlabel('day')
plt.ylabel('Result')
plt.show()
```



By grouping the data by the monitoring program, this code generates a line plot that displays the variation in the "Result" of a particular water quality metric (in this case, "Dissolved Oxygen") across several days. It enables us to see any patterns or trends in the data across time and between various programs for this parameter.

In [28]:

```
# Create a scatterplot of dissolved oxygen vs temperature
sns.scatterplot(data=df, x='Result', y='Result')
plt.title('Relationship Between Dissolved Oxygen and Temperature')
plt.xlabel('Temperature (C)')
plt.ylabel('Dissolved Oxygen (mg/L)')
plt.show()
```



Given that the x and y axes are both set to "Result," there appears to be an error in the code. Without more context, it is unclear what they stand for. However, it appears from the plot title and axis labels that the goal was to make a scatterplot to study the correlation between temperature and dissolved oxygen.

In [29]:

```
# Calculate summary statistics for each parameter
df.groupby('Parameter')['Result'].describe()
```

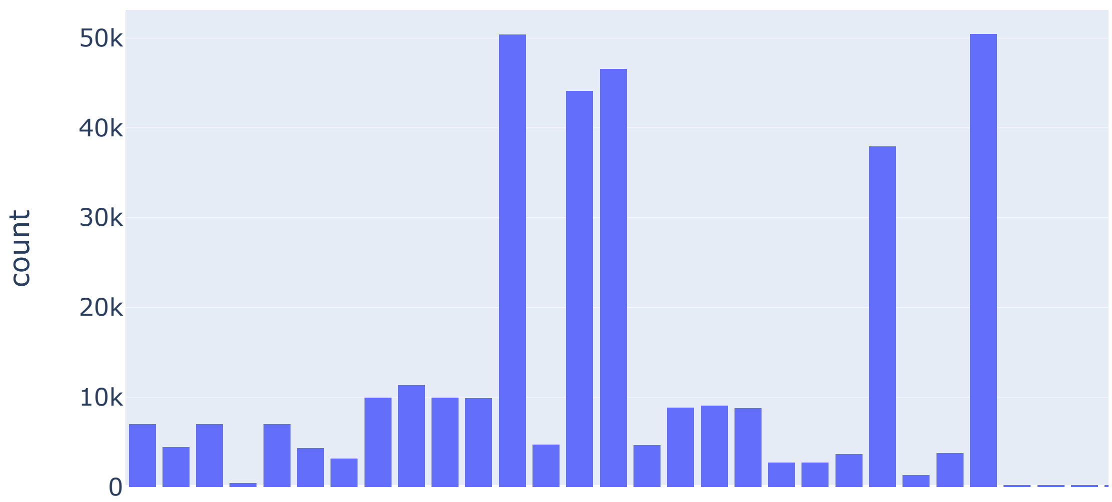
Out[29]:

Parameter	count	unique	top	freq
Ammonia-Nitrogen	4625	860	<0.00439	211
Antimony_Dissolved	457	22	<0.20	375
Antimony_Total	2706	192	<0.20	1880
Arsenic_Dissolved	454	36	<1.00	349
Arsenic_Total	2724	207	<1.00	1595
BOD5	4299	252	<2	1160
COD	9054	500	18	488
Cadmium_Dissolved	616	99	<0.10	273
Cadmium_Total	3127	419	<0.10	1477
Chloride	8841	1879	65	92
Chlorine	4723	16	<0.1	4616
Chlorine_Spec	42656	122	0	17632
Chromium_Dissolved	739	69	<1.00	352
Chromium_Total	3671	455	<1.00	1315
Conductivity	50369	7186	498	214
Copper, Total	19	7	<11.1	13
Copper_Dissolved	1853	373	<2	567
Copper_Total	6994	926	<2	651
Dissolved Oxygen	37927	1658	12.01	171
E. Coli	4107	803	>2419.6	231
Enterococcus	2818	759	>2419.6	235
Fecal Coliform	4438	133	5000	274
Fluoride	8763	1037	<1.0	47
Hach Ammonia-Nitrogen	50440	332	0	8717
Hardness	4506	762	160	137
Iron, Total	5	5	340	1
Lead, Total	8	1	<6.7	8
Lead_Dissolved	1829	235	<2	298
Lead_Total	6993	756	<1.00	914
Maximum Depth	2	2	4	1
Nitrate+Nitrite-Nitrogen	9918	2290	1.2	188
Oil & Grease	344	43	<2	71
Ortho-Phosphate	1304	135	0.01	75
SGT-HEM	128	14	<2	62
Secchi Disk	94	52	4.5	7
Sodium	885	425	29	22

	count	unique	top	freq
Parameter				
Suspended Solids	11316	332	<1	819
TKN	9916	1974	<0.5	934
Total Coliform	3765	83	5000	260
Total Petroleum Hydrocarbons	130	18	<2	43
Total Phenolics	199	10	<20	115
Total Phosphorus	9898	643	0.03	344
Turbidity	10989	1528	2.03	75
Water Temperature	46517	2972	22	112
Zinc, Total	8	7	<22	2
Zinc_Dissolved	1850	459	<10	246
Zinc_Total	6987	2101	<10	447
chlorine	1	1	0.05	1
pH	44094	406	8.01	639

In [31]:

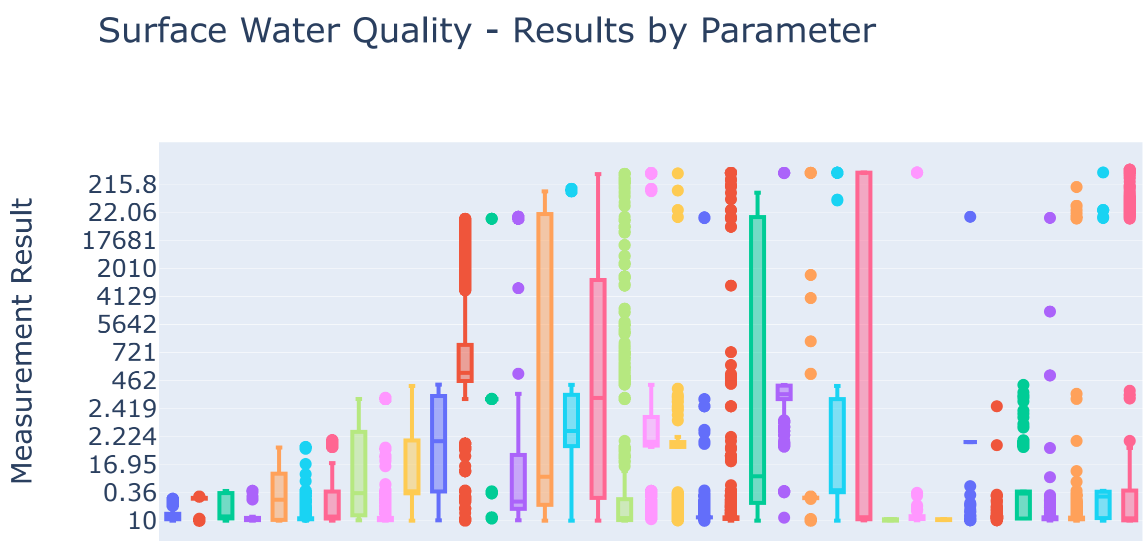
```
#By using Plotly Express library to create a histogram of the 'Parameter' column in the d
import plotly.express as px
fig3 = px.histogram(df, x='Parameter', nbins=20)
fig3.show()
```



The Plotly Express library is used in this code to produce a histogram. The dataframe's 'Parameter' column's value distribution is shown as a histogram with 20 bins. The figure that results displays how frequently each parameter occurs.

In [32]:

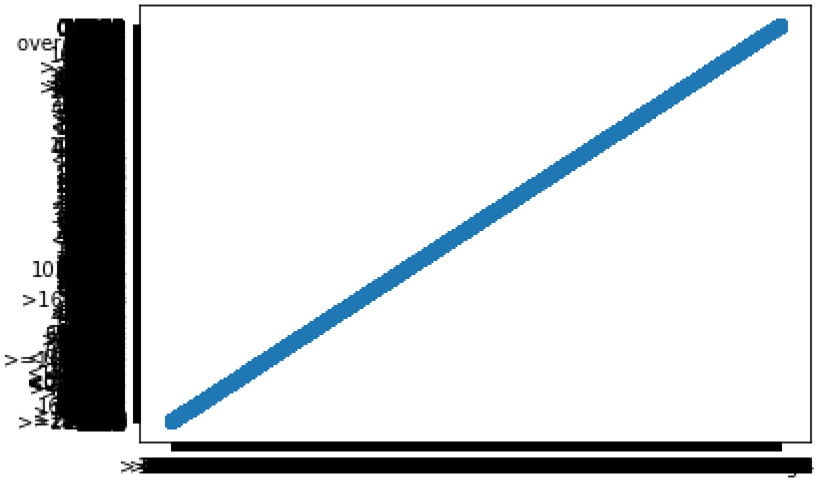
```
fig2 = px.box(df, x='Parameter', y='Result', color='Parameter',
title='Surface Water Quality - Results by Parameter',
labels={'Result': 'Measurement Result', 'Parameter': 'Water Quality Parameter'})
fig2.show()
```



Using the Plotly Express package, this code generates a box plot to display the distribution of measurement data for several water quality indicators. Each box indicates the quartiles of the distribution for a certain parameter, while the x-axis displays the parameters and the y-axis displays the measurement results. The box's color matches the parameter being measured. For clarity, the title and axis labels have also been modified.

In [33]:

```
# Plot the scatter plot of the most important feature and target variable
plt.scatter(df['Result'], df.Result)
plt.show()
```

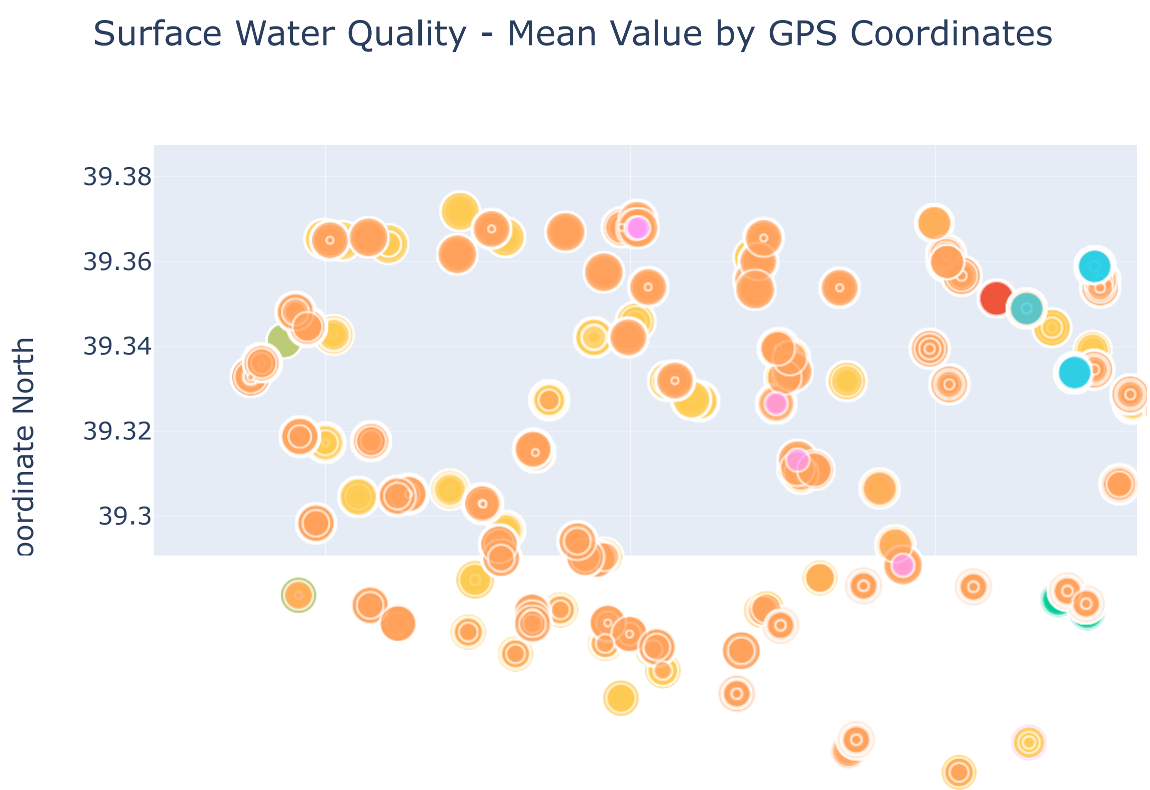


The code is generating a scatter plot of the 'Result' variable against itself, which doesn't provide any useful information. It appears to be an error.

In [34]:

```
# Create a scatter plot of mean value by GPS coordinates
fig = px.scatter(df, x='GPS Coordinate West', y='GPS Coordinate North', color='Parameter
                 size='day', hover_data=['Lab', 'Unit'],
                 title='Surface Water Quality - Mean Value by GPS Coordinates')


fig.show()
```



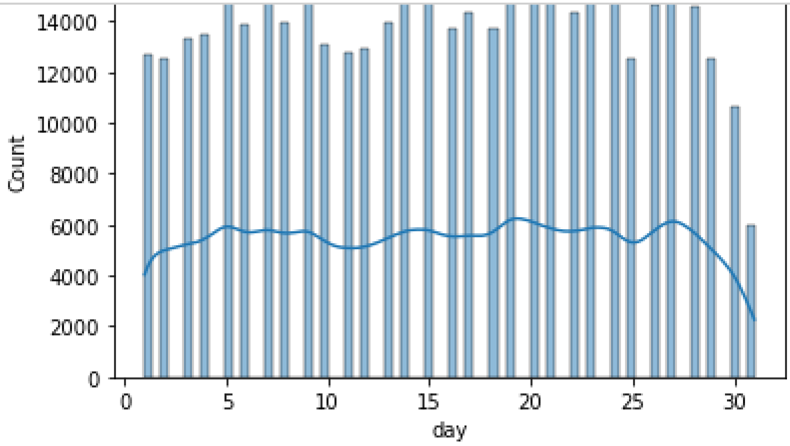
This code creates a scatter plot using Plotly Express (px) library to display the mean value of a water quality parameter for each GPS coordinate in the dataset. The color and size of each point represent the water quality parameter and day, respectively, and additional information about the Lab and Unit are displayed upon hovering over each point. The resulting plot can help identify any spatial patterns or trends in water quality measurements across the study area.

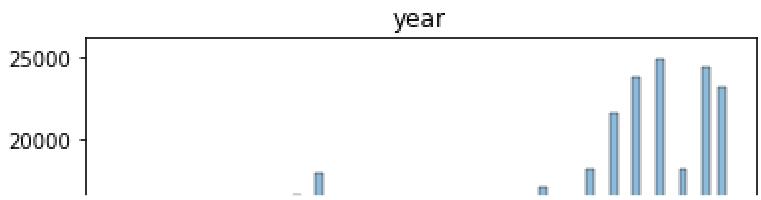
In [36]:

```
import seaborn as sns

# Select numerical columns from the dataframe
numerical_cols = ['GPS Coordinate North', 'GPS Coordinate West', 'Result', 'month', 'day

# Plot histograms for each column
for col in numerical_cols:
    plt.figure()
    sns.histplot(data=df, x=col, kde=True)
    plt.title(col)
    plt.show()
```



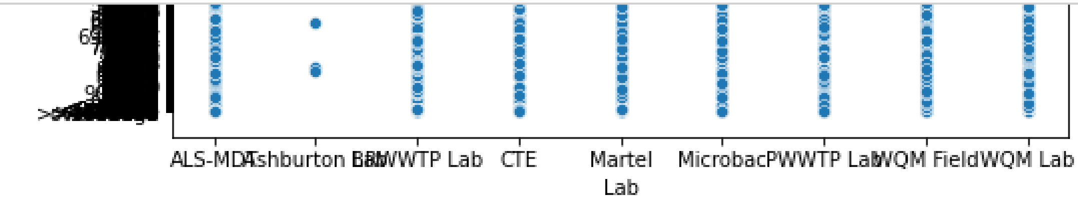


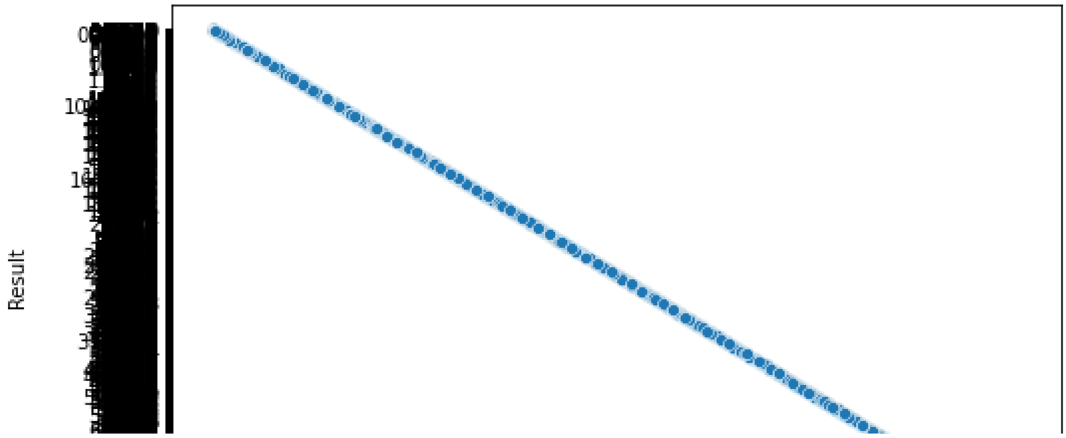
The histplot function from Seaborn is used in this code to pick the dataframe's numerical columns and plot histograms for each of them. It generates a unique histogram graphic for each column by iteratively traversing each column. The histogram's kernel density estimation line is added when the kde=True option is set. Each plot's title is prefixed with the column's name. This code's objective is to display the value distribution across each numerical column.

In [37]:

```
# Detect outliers using box plots
plt.figure(figsize=(15,8))
sns.boxplot(data=df.drop(['Station'], axis=1), orient="h")
plt.show()

# Detect outliers using scatter plots
for attribute in df.drop(['Station'], axis=1).columns:
    plt.figure(figsize=(8, 5))
    sns.scatterplot(x=df[attribute], y=df['Result'])
    plt.show()
```





Box plots and scatter plots are the two methods for outlier detection that the code offers.

The first graph displays the distribution of each numerical variable in the dataset as a horizontal box plot. The whiskers extend to the minimum and highest values within 1.5 times the interquartile range (IQR), which is shown by the box. An outlier is any point that lies outside the whiskers.

Each numerical variable in the dataset is plotted against the 'Result' variable in the second plot, which is a scatter plot. Visually, outliers are identified as points that are spatially far from the rest of the points.

Conclusion

The distribution of each characteristic, the relationships between features, and the existence of outliers are all fully understood by using these visualization and analytic approaches on the water quality dataset. Making decisions regarding the data, such as which characteristics to include in a model and how to manage outliers, may be done with the use of this information.

In [ ]: