Introduction

This assignment involves analyzing a dataset derived from an open USGS dataset for water quality monitoring at the Port of Albany, New York, on the Hudson River. The dataset, sourced from the USGS Station Website, includes various water quality parameters such as water temperature, dissolved oxygen, pH, turbidity, and more. The data have been minimally processed and quality controlled (QC'd) to provide an accurate foundation for analysis.

The objective of this assignment is to explore the dataset, address any potential data quality issues, and apply data science techniques to extract meaningful insights. Specifically, this analysis focuses on documenting the development tools used (Task 1), performing exploratory data analysis (Task 2), identifying interesting characteristics of the data (Task 3), and developing a predictive model (Task 5).

The outcome of this assignment will not only demonstrate proficiency in handling environmental datasets but also showcase the practical application of data science in addressing water quality challenges.

Task 1 - Documenting Development Tools

Operating System

- **Microsoft Windows 10 Home, 64-bit**
- **Version**: 10.0.19044 Build 19044

Programming Language

- **Python 3.12.6**

Tools

- **Jupyter Notebook**: For interactive coding.
- **Git**: For version control and repository management.

Data Files

- `dev.csv` : The main dataset used for analysis and model development.
- `test.csv` : Dataset used for testing and model validation.

Version Control

- The repository was forked from gcoyle83/ds-interviews.
- All work was version-controlled using Git, with changes committed regularly and pushed to the forked repository.

Environment Management

- A virtual environment was created using `virtualenv` to ensure that all project dependencies are isolated from the global environment, preventing conflicts with other projects and ensuring consistency.

- The virtual environment (`myenv`) was activated, and the necessary packages for the project were installed, including:

    - **Jupyter Notebook**: for creating and running interactive Python notebooks.
    - **pandas**: for data manipulation and analysis, particularly useful for working with CSV files.
    - **matplotlib**: for creating static, animated, and interactive visualizations in Python.
    - **seaborn**: for creating aesthetically pleasing statistical graphics.
    - **scikit-learn**: for machine learning model implementation.
    - **scipy**: for scientific computing and statistics.
    - **statsmodels**: for statistical modeling and hypothesis testing.
- To ensure reproducibility, a `requirements.txt` file was generated, which lists all the project dependencies along with their specific versions. This allows anyone working with the project to recreate the exact same environment.

- The project dependencies can be installed via:

  ```
  pip install -r requirements.txt
  ```

Task 2 - Describe the development data set

For Task 2, an exploratory data analysis was performed on the development dataset. Initially, summary statistics such as count, mean, standard deviation, and various percentiles (min, max, 25th, 50th, and 75th) were calculated for each feature. This provided insights into the central tendencies and distribution characteristics of different environmental variables.

In addition to summary statistics, a completeness analysis was conducted by calculating the number of missing and non-missing values for each feature. These statistics were compiled into a table and further visualized using a bar plot that highlighted the percentage of missing data for each feature. A color-coding system was applied to distinguish different levels of missingness: features with more than 1% missing data were shown in red, those with 0.5% to 1% missing were shown in gold, and features with less than 0.5% missing data were marked in green.

Finally, a set of Sina plots was created for each feature, visually displaying the distribution along with key statistics such as min, 1st quartile, median, mean, 3rd quartile, and max. These plots offered a more detailed view of the spread and variability of the data, helping inform decisions about preprocessing and feature engineering in subsequent modeling tasks.

In [ ]:

```python
import pandas as pd
# Load the dataset
dev_df = pd.read_csv('dev.csv')

# Display the first few rows
dev_df.head()
```

In [4]:

```python
# Generate summary statistics for all numerical columns
empirical_stats = dev_df.describe()

# Display the statistics
empirical_stats
```

Out[4]:

	wtempc	atempc	winddir_dcfn	precp_in	relh_pct	spc	dox_mgl
count	39408.000000	39548.000000	39548.000000	39548.000000	39551.000000	39401.000000	39390.000000
mean	11.273574	10.180927	160.069030	0.000727	72.246110	238.469760	11.857111
std	8.958956	9.622149	106.167885	0.005529	19.596247	40.555304	2.908125
min	0.000000	-23.500000	0.000000	0.000000	13.100000	140.000000	6.900000
25%	2.700000	2.200000	48.000000	0.000000	58.200000	210.000000	8.800000
50%	9.600000	9.000000	183.000000	0.000000	73.900000	241.000000	12.100000
75%	21.000000	18.800000	199.000000	0.000000	89.000000	262.000000	14.600000
max	26.500000	33.300000	360.000000	0.200000	100.000000	420.000000	16.500000

In [93]:

```python
import pandas as pd

# Assuming `dev_df` is your DataFrame
# Define the descriptions and units for each feature
feature_info = {
    'wtempc': 'Water Temperature (°C)',
    'atempc': 'Air Temperature (°C)',
    'winddir_dcfn': 'Wind Direction (°)',
    'precp_in': 'Precipitation (in)',
    'relh_pct': 'Relative Humidity (%)',
    'spc': 'Specific Conductivity (µS/cm)',
    'dox_mgl': 'Dissolved Oxygen (mg/L)',
    'ph': 'pH (0-14 SU)',
    'windgust_knots': 'Wind Gust Speed (knots)',
    'wse1988': 'Water Surface Elevation (NAVD88)',
    'wvel_fps': 'Water Velocity (ft/s)',
    'mbars': 'Atmospheric Pressure (millibars)',
    'windspeed_knots': 'Wind Speed (knots)',
    'par': 'Photosynthetically Available Radiation (mmol/m²/s)',
    'turb_fnu': 'Turbidity (FNU)',
    'datetime_utc': 'Timestamp (UTC)'  # Add any additional features that are present in dev_df
}

# Count missing values for each feature
missing_values = dev_df.isnull().sum()

# Count non-missing values for each feature
non_missing_values = dev_df.notnull().sum()

# Calculate percentages
missing_percentage = (missing_values / len(dev_df)) * 100
non_missing_percentage = (non_missing_values / len(dev_df)) * 100

# Map feature names to their descriptions for the table
```

```
descriptions = [feature_info.get(feature, feature) for feature in dev_df.columns]

# Create a DataFrame summarizing completeness with percentages and descriptions
completeness_summary = pd.DataFrame({
    'Feature Description': descriptions,
    'Missing Values': missing_values,
    'Non-Missing Values': non_missing_values,
    'Missing Percentage (%)': missing_percentage,
    'Non-Missing Percentage (%)': non_missing_percentage,
    'Total Records': len(dev_df)
}, index=dev_df.columns)

# Display the completeness summary with feature descriptions
completeness_summary
```

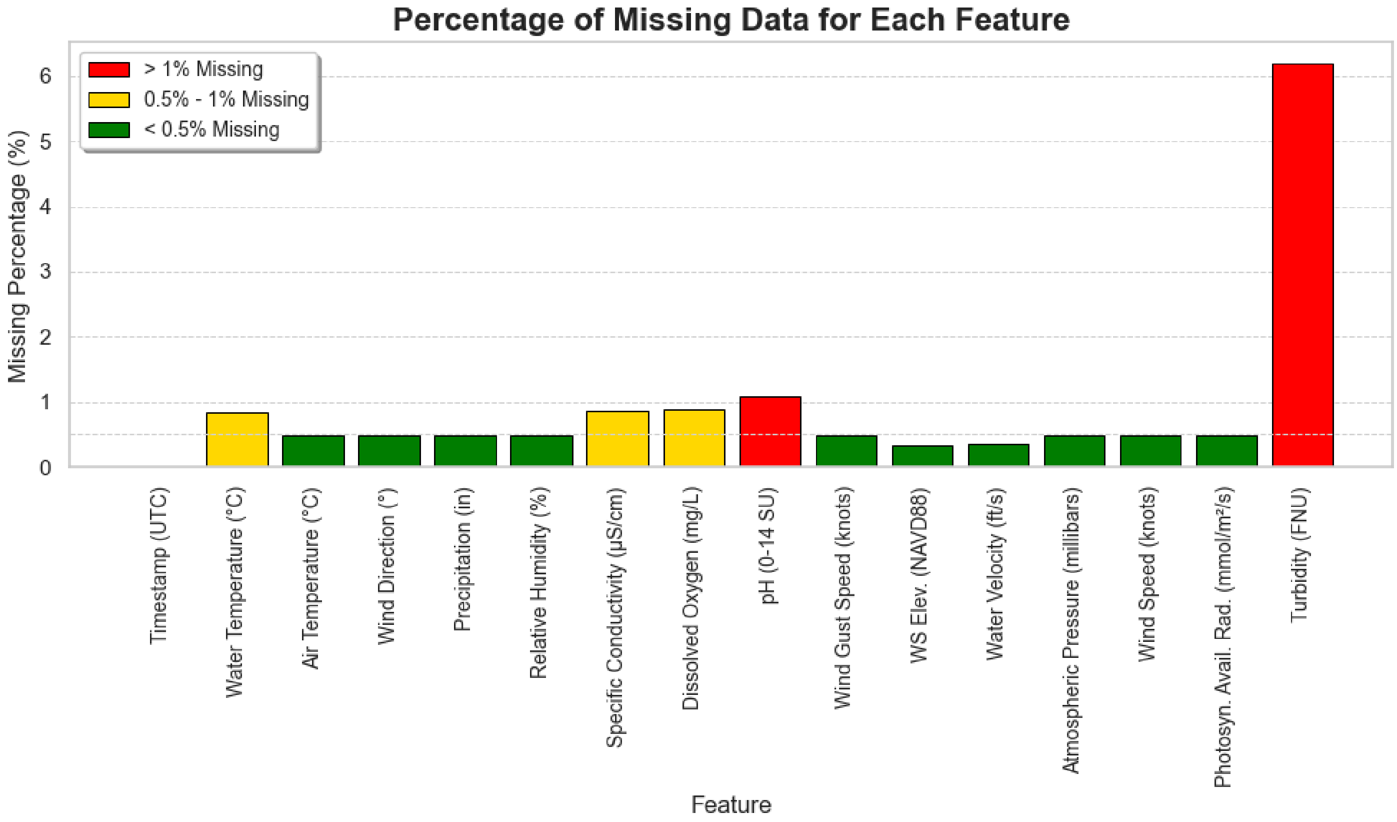
Out[93]:

	Feature Description	Missing Values	Non-Missing Values	Missing Percentage (%)	Non-Missing Percentage (%)	Total Records
datetime_utc	Timestamp (UTC)	0	39744	0.000000	100.000000	39744
wtempc	Water Temperature (°C)	336	39408	0.845411	99.154589	39744
atempc	Air Temperature (°C)	196	39548	0.493156	99.506844	39744
winddir_dcfn	Wind Direction (°)	196	39548	0.493156	99.506844	39744
precp_in	Precipitation (in)	196	39548	0.493156	99.506844	39744
relh_pct	Relative Humidity (%)	193	39551	0.485608	99.514392	39744
spc	Specific Conductivity (µS/cm)	343	39401	0.863023	99.136977	39744
dox_mgl	Dissolved Oxygen (mg/L)	354	39390	0.890700	99.109300	39744
ph	pH (0-14 SU)	431	39313	1.084440	98.915560	39744
windgust_knots	Wind Gust Speed (knots)	193	39551	0.485608	99.514392	39744
wse1988	Water Surface Elevation (NAVD88)	134	39610	0.337158	99.662842	39744
wvel_fps	Water Velocity (ft/s)	144	39600	0.362319	99.637681	39744
mbars	Atmospheric Pressure (millibars)	196	39548	0.493156	99.506844	39744
windspeed_knots	Wind Speed (knots)	193	39551	0.485608	99.514392	39744
par	Photosynthetically Available Radiation (mmol/m...	196	39548	0.493156	99.506844	39744
turb_fnu	Turbidity (FNU)	2465	37279	6.202194	93.797806	39744

In [92]:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.patches import Patch
```

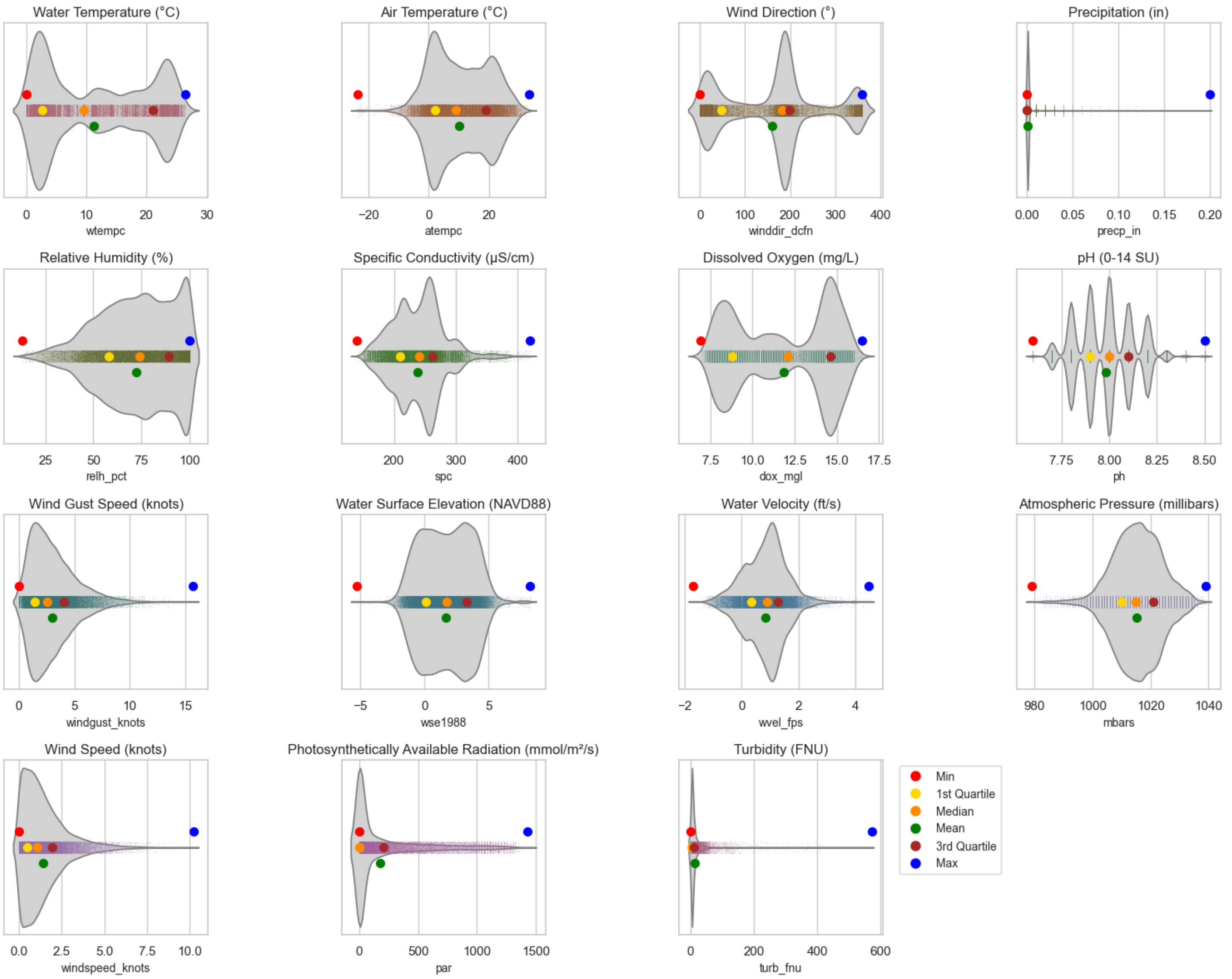
```python
# Define the descriptions and units for each feature
feature_info = {
    'wtempc': 'Water Temperature (°C)',
    'atempc': 'Air Temperature (°C)',
    'winddir_dcfn': 'Wind Direction (°)',
    'precp_in': 'Precipitation (in)',
    'relh_pct': 'Relative Humidity (%)',
    'spc': 'Specific Conductivity (µS/cm)',
    'dox_mgl': 'Dissolved Oxygen (mg/L)',
    'ph': 'pH (0-14 SU)',
    'windgust_knots': 'Wind Gust Speed (knots)',
    'wse1988': 'WS Elev. (NAVD88)',
    'wvel_fps': 'Water Velocity (ft/s)',
    'mbars': 'Atmospheric Pressure (millibars)',
    'windspeed_knots': 'Wind Speed (knots)',
    'par': 'Photosyn. Avail. Rad. (mmol/m²/s)',
    'turb_fnu': 'Turbidity (FNU)',
    'datetime_utc': 'Timestamp (UTC)'  # Added to avoid KeyError
}

# Map feature acronyms to their descriptions, using the feature name as a default if not found
x_labels = [feature_info.get(feature, feature) for feature in missing_percentage.index]

# Define colors based on missing percentages
colors = []
for percent in missing_percentage:
    if percent > 1:
        colors.append('red')
    elif 0.5 <= percent <= 1:
        colors.append('gold')
    else:
        colors.append('green')

# Create the figure and axes
fig, ax = plt.subplots(figsize=(10, 6))

# Create the bar plot with edge color for better definition
bars = ax.bar(x_labels, missing_percentage, color=colors, edgecolor='black', linewidth=0.7)

# Add horizontal gridlines
ax.yaxis.grid(True, color='lightgrey', linestyle='--', linewidth=0.7)
ax.xaxis.grid(False)  # Remove vertical gridlines for cleaner look

# Add a gridline for 0.5% to highlight the threshold
plt.axhline(y=0.5, color='lightgrey', linestyle='--', linewidth=0.7)

# Set plot titles and labels
plt.title('Percentage of Missing Data for Each Feature', fontsize=16, fontweight='bold')
plt.ylabel('Missing Percentage (%)', fontsize=12)
plt.xlabel('Feature', fontsize=12)
plt.xticks(rotation=90, fontsize=10)  # Adjust font size for better readability

# Add color-coded legend with custom styling
legend_elements = [
    Patch(facecolor='red', edgecolor='black', linewidth=0.7, label='> 1% Missing'),
    Patch(facecolor='gold', edgecolor='black', linewidth=0.7, label='0.5% - 1% Missing'),
    Patch(facecolor='green', edgecolor='black', linewidth=0.7, label='< 0.5% Missing')
]
ax.legend(handles=legend_elements, loc='upper left', fontsize=10, frameon=True, fancybox=True, s
# Enhance plot layout for paper-quality output
plt.tight_layout()

# Show the final plot
plt.show()
```



In [94]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Set plot style for a cleaner look
sns.set(style="whitegrid")

# Define the descriptions and units for each feature
feature_info = {
    'wtempc': 'Water Temperature (°C)',
    'atempc': 'Air Temperature (°C)',
    'winddir_dcfn': 'Wind Direction (°)',
    'precp_in': 'Precipitation (in)',
    'relh_pct': 'Relative Humidity (%)',
    'spc': 'Specific Conductivity (µS/cm)',
    'dox_mgl': 'Dissolved Oxygen (mg/L)',
    'ph': 'pH (0-14 SU)',
    'windgust_knots': 'Wind Gust Speed (knots)',
    'wse1988': 'Water Surface Elevation (NAVD88)',
    'wvel_fps': 'Water Velocity (ft/s)',
    'mbars': 'Atmospheric Pressure (millibars)',
    'windspeed_knots': 'Wind Speed (knots)',
    'par': 'Photosynthetically Available Radiation (mmol/m²/s)',
    'turb_fnu': 'Turbidity (FNU)'
}

# Select numerical columns from dev_df for sina plot
```

```
numerical_columns = dev_df.select_dtypes(include='number').columns

# Define a color palette for each feature
colors = sns.color_palette("husl", len(numerical_columns))  # husl generates distinct colors for

# Set the figure size to provide more space for each plot
plt.figure(figsize=(15, 12))

# Create sina plots with statistical markers
for i, col in enumerate(numerical_columns, 1):
    plt.subplot(4, 4, i)

    # Background density with a violin plot (light grey)
    sns.violinplot(x=dev_df[col], inner=None, color="lightgrey")

    # Sina plot effect using jittered strip plot with smaller, semi-transparent points
    sns.stripplot(x=dev_df[col], jitter=0.03, size=0.3, color=colors[i-1], alpha=0.5)

    # Calculate the statistical values
    min_val = dev_df[col].min()
    max_val = dev_df[col].max()
    q1 = dev_df[col].quantile(0.25)
    median = dev_df[col].median()
    mean = dev_df[col].mean()
    q3 = dev_df[col].quantile(0.75)

    # Overlay larger dots for statistical values
    plt.scatter([min_val], [-0.08], color='red', s=50, label='Min', zorder=3)      # Min value (
    plt.scatter([q1], [0], color='gold', s=50, label='1st Quartile', zorder=3)     # 1st quartil
    plt.scatter([median], [0], color='darkorange', s=50, label='Median', zorder=3) # Median
    plt.scatter([mean], [0.08], color='green', s=50, label='Mean', zorder=3)       # Mean (above
    plt.scatter([q3], [0], color='brown', s=50, label='3rd Quartile', zorder=3)    # 3rd quartil
    plt.scatter([max_val], [-0.08], color='blue', s=50, label='Max', zorder=3)     # Max value

    # Use the feature descriptions and units from the dictionary for the title
    plt.title(feature_info.get(col, col), fontsize=12)  # Use description if available, else col
    plt.xlabel(col, fontsize=10)

# Create a legend for the markers
handles = [
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='red', markersize=10, label='Min
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='gold', markersize=10, label='1s
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='darkorange', markersize=10, lab
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='green', markersize=10, label='M
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='brown', markersize=10, label='3
    plt.Line2D([0], [0], marker='o', color='w', markerfacecolor='blue', markersize=10, label='Ma
]
plt.legend(handles=handles, loc='upper left', bbox_to_anchor=(1.05, 1), fontsize=10)

# Tight layout to ensure plots do not overlap
plt.tight_layout()
plt.show()
```



Task 3 - Explore Interesting Characteristics of the Dataset

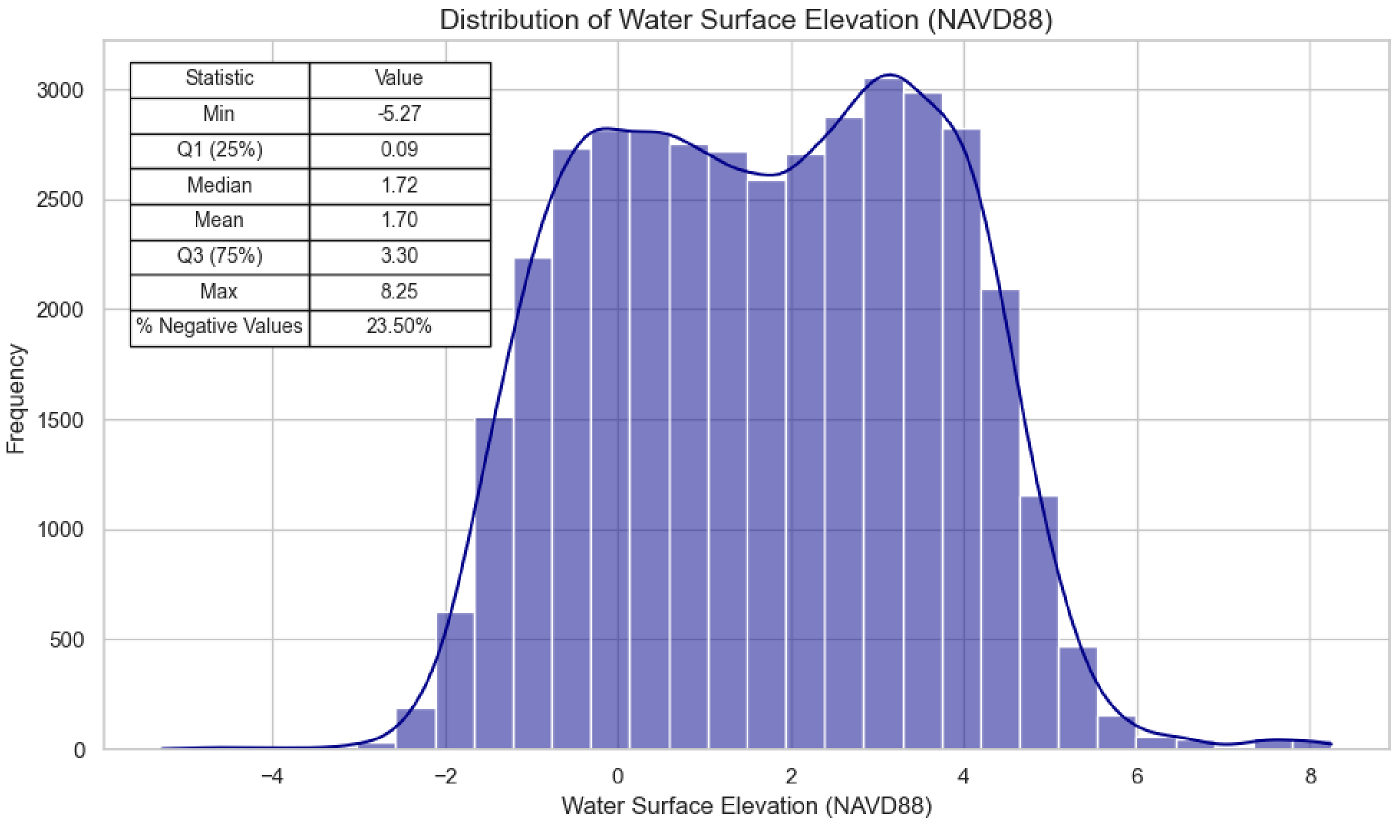
Distribution of Water Surface Elevation (NAVD88)

The plot shows the distribution of water surface elevation values, referenced to the NAVD88 datum. The data exhibits a roughly **normal distribution**, centered around **0 to 3 feet**, but with notable **negative values**. The presence of negative water surface elevation values indicates that in some cases, the water level fell below the chosen reference datum (NAVD88). This could be due to topographical factors, seasonal fluctuations, or variations in the water body's management, such as reservoir drawdowns. On the upper end, the distribution tapers off around **8 feet**, indicating that higher water surface elevations are less frequent. Overall, the data suggests a wide variability in water levels, which may need to be carefully handled in hydrological modeling.

In [82]:

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

# Calculate summary statistics for water surface elevation
min_val = dev_df['wse1988'].min()
```

```python
q1 = dev_df['wse1988'].quantile(0.25)
median = dev_df['wse1988'].median()
mean_val = dev_df['wse1988'].mean()
q3 = dev_df['wse1988'].quantile(0.75)
max_val = dev_df['wse1988'].max()

# Calculate the percentage of negative values
negative_count = (dev_df['wse1988'] < 0).sum()
negative_percentage = (negative_count / len(dev_df)) * 100

# Create the plot
plt.figure(figsize=(10, 6))
sns.histplot(dev_df['wse1988'], bins=30, kde=True, color='darkblue')

plt.title('Distribution of Water Surface Elevation (NAVD88)', fontsize=14)
plt.xlabel('Water Surface Elevation (NAVD88)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)

# Create a small table with summary statistics
table_data = [
    ["Min", f"{min_val:.2f}"],
    ["Q1 (25%)", f"{q1:.2f}"],
    ["Median", f"{median:.2f}"],
    ["Mean", f"{mean_val:.2f}"],
    ["Q3 (75%)", f"{q3:.2f}"],
    ["Max", f"{max_val:.2f}"],
    ["% Negative Values", f"{negative_percentage:.2f}%"]
]

# Add the table to the plot (positioned inside the plot), adjusting the position
table = plt.table(cellText=table_data, colWidths=[0.1, 0.1], cellLoc='center', rowLoc='center',
                  colLabels=["Statistic", "Value"], loc="upper left", bbox=[0.02, 0.57, 0.28, 0.4

# Adjust cell height by increasing the height of each row
table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(0.8, 1.2)  # Increase the row height by scaling the table vertically

# Set the table background to white to avoid gridlines overlaying it
table.set_zorder(10)  # Ensure the table is on top of the plot
table_props = table.properties()
table_cells = table_props['children']
for cell in table_cells:
    cell.set_facecolor("white")  # Set white background for each cell

# Show the plot
plt.tight_layout()
plt.show()
```



Statistic	Value
Min	-5.27
Q1 (25%)	0.09
Median	1.72
Mean	1.70
Q3 (75%)	3.30
Max	8.25
% Negative Values	23.50%

Distribution of Precipitation (inches)

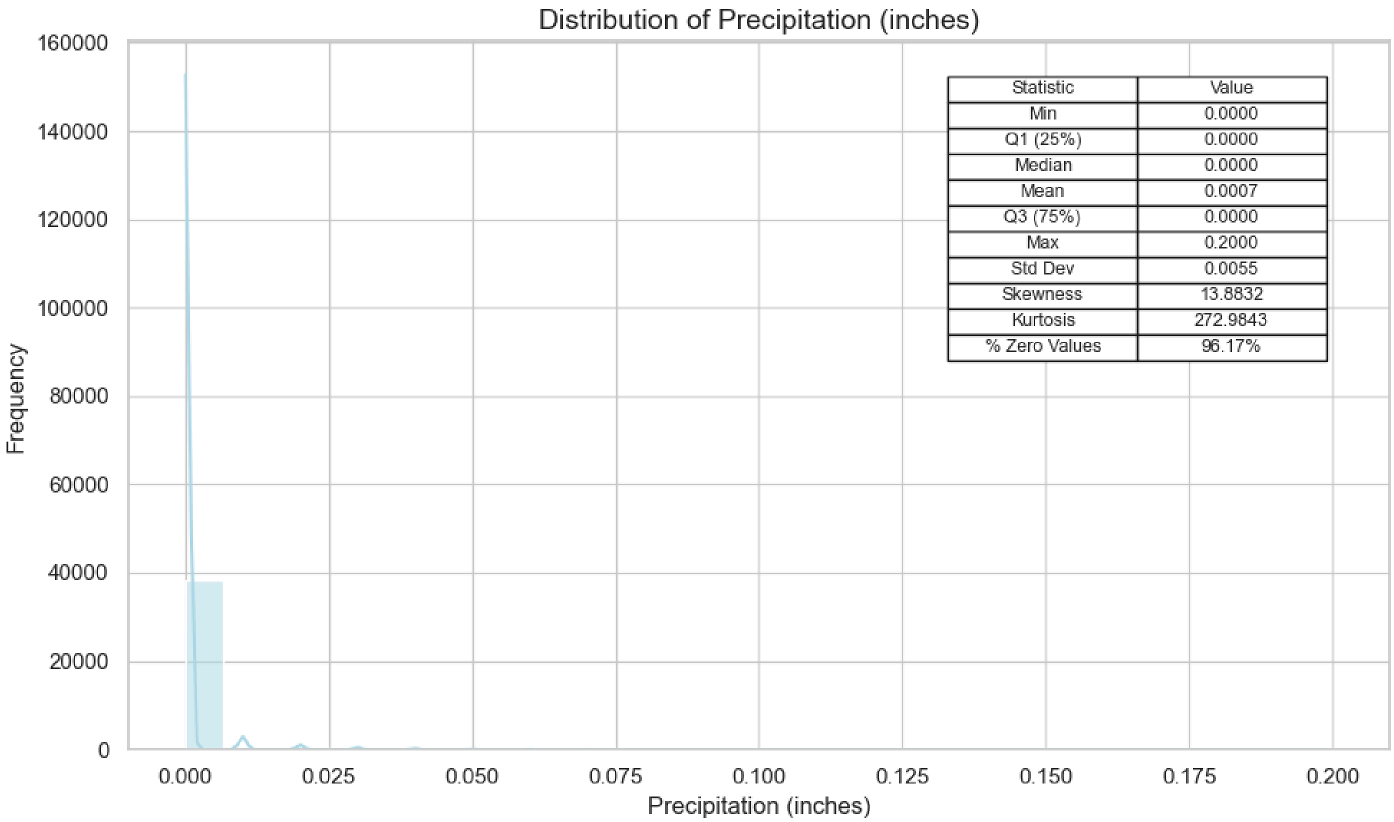
The following plot shows the distribution of precipitation values in inches. A significant observation is that **96.17% of the values** are **zero**, indicating that most of the data points correspond to periods with no recorded precipitation. This heavy concentration of zero values suggests either **long dry spells** in the region, or the possibility that **precipitation data** was not consistently recorded during some periods.

The distribution also shows that when precipitation does occur, the amounts are generally low, with the majority of values falling below **0.05 inches**. This may present challenges in modeling, as the prevalence of zero values could lead to a skewed model unless properly addressed, such as by using specialized techniques for handling zero-inflated data.

In [81]:

```
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats

# Calculate the percentage of zero precipitation values
zero_precip_count = (dev_df['precp_in'] == 0).sum()
zero_precip_percentage = (zero_precip_count / len(dev_df)) * 100

# Calculate summary statistics for precipitation
min_val = dev_df['precp_in'].min()
q1 = dev_df['precp_in'].quantile(0.25)
median = dev_df['precp_in'].median()
mean_val = dev_df['precp_in'].mean()
q3 = dev_df['precp_in'].quantile(0.75)
max_val = dev_df['precp_in'].max()
std_val = dev_df['precp_in'].std()
skewness = dev_df['precp_in'].skew()
```

```python
kurtosis = dev_df['precp_in'].kurtosis()

# Create the plot
plt.figure(figsize=(10, 6))
sns.histplot(dev_df['precp_in'], bins=30, kde=True, color='lightblue')

plt.title('Distribution of Precipitation (inches)', fontsize=14)
plt.xlabel('Precipitation (inches)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)

# Create a small table with summary statistics and zero percentage
table_data = [
    ["Min", f"{min_val:.4f}"],  # More precision for precipitation values
    ["Q1 (25%)", f"{q1:.4f}"],
    ["Median", f"{median:.4f}"],
    ["Mean", f"{mean_val:.4f}"],
    ["Q3 (75%)", f"{q3:.4f}"],
    ["Max", f"{max_val:.4f}"],
    ["Std Dev", f"{std_val:.4f}"],
    ["Skewness", f"{skewness:.4f}"],
    ["Kurtosis", f"{kurtosis:.4f}"],
    ["% Zero Values", f"{zero_precip_percentage:.2f}%"]
]

# Add the table to the plot (positioned at the top-right), adjusting the position
table = plt.table(cellText=table_data, colWidths=[0.3, 0.3], cellLoc='center', rowLoc='center',
                  colLabels=["Statistic", "Value"], loc="upper right", bbox=[0.65, 0.55, 0.3, 0.4

# Adjust the table size
table.auto_set_font_size(False)
table.set_fontsize(9)  # Keep the font size slightly smaller
table.scale(1.2, 1.5)  # Scale the table to increase row width and height

# Set the table background to white to avoid gridlines overlaying it
table.set_zorder(10)  # Ensure the table is on top of the plot
table_props = table.properties()
table_cells = table_props['children']
for cell in table_cells:
    cell.set_facecolor("white")

# Show the plot
plt.tight_layout()
plt.show()
```



Statistic	Value
Min	0.0000
Q1 (25%)	0.0000
Median	0.0000
Mean	0.0007
Q3 (75%)	0.0000
Max	0.2000
Std Dev	0.0055
Skewness	13.8832
Kurtosis	272.9843
% Zero Values	96.17%

Distribution of Turbidity (FNU)

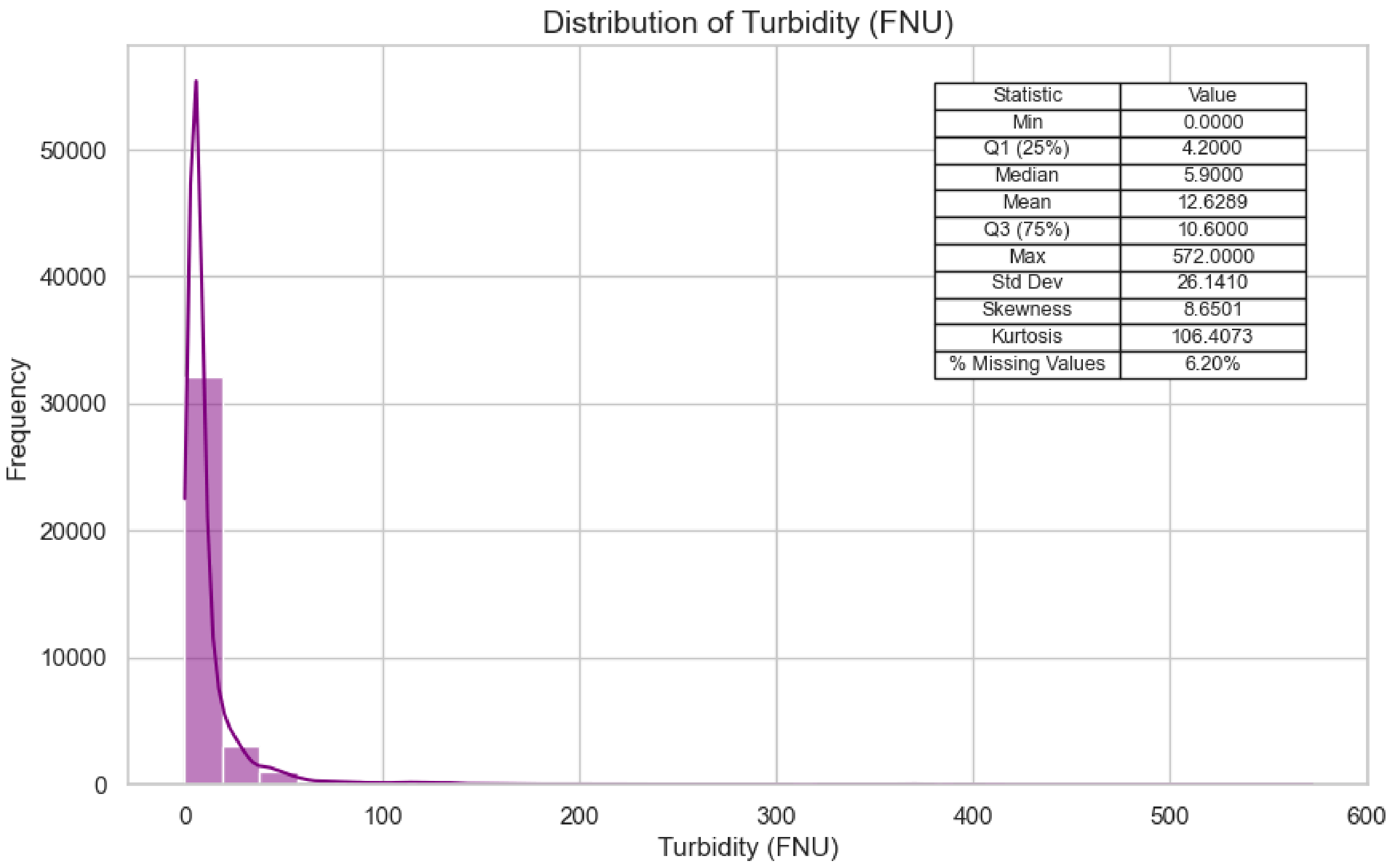
The plot below illustrates the distribution of turbidity values, measured in **Formazin Nephelometric Units (FNU)**. The distribution is highly **right-skewed**, with the majority of values concentrated near **0 FNU**. This suggests that in most cases, the water has very low turbidity, indicating clear water. However, there are a few instances where turbidity levels exceed **100 FNU**, with some extreme values reaching above **400 FNU**.

These higher turbidity values could be the result of **specific environmental events** such as heavy rainfall, runoff, or disturbances in the water body. The extreme skewness in the distribution may present challenges for modeling, as the wide range and extreme values could impact the performance of models unless properly accounted for. Handling the outliers or applying transformations might be necessary to improve model performance.

In [83]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats

# Calculate the percentage of missing turbidity values
missing_turbidity = dev_df['turb_fnu'].isnull().mean() * 100

# Calculate summary statistics for turbidity
min_val = dev_df['turb_fnu'].min()
q1 = dev_df['turb_fnu'].quantile(0.25)
median = dev_df['turb_fnu'].median()
mean_val = dev_df['turb_fnu'].mean()
q3 = dev_df['turb_fnu'].quantile(0.75)
max_val = dev_df['turb_fnu'].max()
std_val = dev_df['turb_fnu'].std()
skewness = dev_df['turb_fnu'].skew()
```

```python
kurtosis = dev_df['turb_fnu'].kurtosis()

# Create the plot
plt.figure(figsize=(10, 6))
sns.histplot(dev_df['turb_fnu'], bins=30, kde=True, color='purple')

# Update the plot title
plt.title('Distribution of Turbidity (FNU)', fontsize=14)
plt.xlabel('Turbidity (FNU)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)

# Create a small table with summary statistics and missing percentage
table_data = [
    ["Min", f"{min_val:.4f}"],  # More precision for turbidity values
    ["Q1 (25%)", f"{q1:.4f}"],
    ["Median", f"{median:.4f}"],
    ["Mean", f"{mean_val:.4f}"],
    ["Q3 (75%)", f"{q3:.4f}"],
    ["Max", f"{max_val:.4f}"],
    ["Std Dev", f"{std_val:.4f}"],
    ["Skewness", f"{skewness:.4f}"],
    ["Kurtosis", f"{kurtosis:.4f}"],
    ["% Missing Values", f"{missing_turbidity:.2f}%"]
]

# Add the table to the plot (positioned at the top-right), adjusting the position
table = plt.table(cellText=table_data, colWidths=[0.3, 0.3], cellLoc='center', rowLoc='center',
                  colLabels=["Statistic", "Value"], loc="upper right", bbox=[0.65, 0.55, 0.3, 0.

# Adjust the table size (increase the height of the cells)
table.auto_set_font_size(False)
table.set_fontsize(9)
table.scale(1.2, 1.5)  # Scale the table to increase row width and height

# Set the table background to white to avoid gridlines overlaying it
table.set_zorder(10)
```



Statistic	Value
Min	0.0000
Q1 (25%)	4.2000
Median	5.9000
Mean	12.6289
Q3 (75%)	10.6000
Max	572.0000
Std Dev	26.1410
Skewness	8.6501
Kurtosis	106.4073
% Missing Values	6.20%

Distribution of pH (SU)

The plot above shows the distribution of pH values, measured in **Standard Units (SU)**, ranging from **7.6 to 8.5**. The data exhibits a **multimodal distribution**, with distinct peaks at several intervals, particularly around **8.0 SU**. These pH values suggest that the water is mostly **neutral to slightly basic**, which is typical for many natural water bodies.

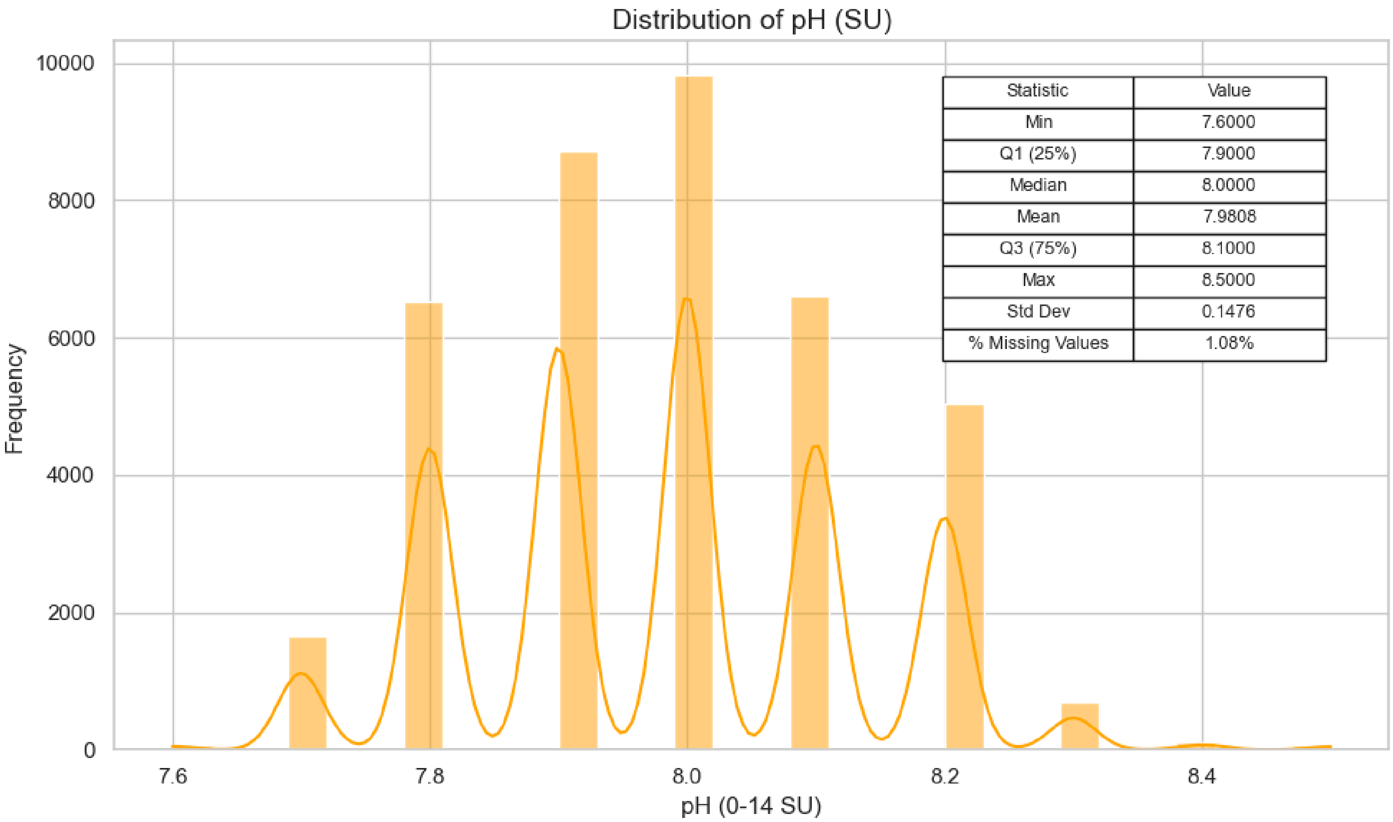
It is also important to note that **1.08% of the pH values are missing**. While the missing value rate is relatively low, it still needs to be addressed in the modeling phase to ensure the integrity of the data. Missing pH values can potentially affect water quality models, especially those sensitive to pH fluctuations, and should be either imputed or handled appropriately.

The narrow range of pH values and the clear peaks could be indicative of stable water chemistry conditions in this environment, but this consistency might also mean that pH might not be a strong differentiator in predictive models without further transformations or feature engineering.

In [88]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Calculate the percentage of missing pH values
missing_ph = dev_df['ph'].isnull().mean() * 100

# Calculate summary statistics for pH
min_val = dev_df['ph'].min()
q1 = dev_df['ph'].quantile(0.25)
median = dev_df['ph'].median()
mean_val = dev_df['ph'].mean()
q3 = dev_df['ph'].quantile(0.75)
```

```python
max_val = dev_df['ph'].max()
std_val = dev_df['ph'].std()

# Create the plot
plt.figure(figsize=(10, 6))
sns.histplot(dev_df['ph'], bins=30, kde=True, color='orange')

# Update the plot title
plt.title('Distribution of pH (SU)', fontsize=14)
plt.xlabel('pH (0-14 SU)', fontsize=12)
plt.ylabel('Frequency', fontsize=12)

# Create a small table with summary statistics and missing percentage
table_data = [
    ["Min", f"{min_val:.4f}"],
    ["Q1 (25%)", f"{q1:.4f}"],
    ["Median", f"{median:.4f}"],
    ["Mean", f"{mean_val:.4f}"],
    ["Q3 (75%)", f"{q3:.4f}"],
    ["Max", f"{max_val:.4f}"],
    ["Std Dev", f"{std_val:.4f}"],
    ["% Missing Values", f"{missing_ph:.2f}%"]
]

# Add the table to the plot (positioned at the top-right), adjusting the position
table = plt.table(cellText=table_data, colWidths=[0.3, 0.3], cellLoc='center', rowLoc='center',
                  colLabels=["Statistic", "Value"], loc="upper right", bbox=[0.65, 0.55, 0.3, 0.4

# Adjust the table size (increase the height of the cells)
table.auto_set_font_size(False)
table.set_fontsize(9)
table.scale(1.2, 1.5)  # Scale the table to increase row width and height

# Set the table background to white to avoid gridlines overlaying it
table.set_zorder(10)  # Ensure the table is on top of the plot
table_props = table.properties()
table_cells = table_props['children']
for cell in table_cells:
    cell.set_facecolor("white")

# Show the plot
plt.tight_layout()
plt.show()
```



Statistic	Value
Min	7.6000
Q1 (25%)	7.9000
Median	8.0000
Mean	7.9808
Q3 (75%)	8.1000
Max	8.5000
Std Dev	0.1476
% Missing Values	1.08%

Task 5 - Model Development

In this task, a predictive model was developed using the **Random Forest Regressor** to estimate air temperature (`atempc`) based on other environmental variables such as **water temperature**, **relative humidity**, **wind speed**, and **atmospheric pressure**. The Random Forest algorithm was chosen for its flexibility, ability to handle non-linear relationships, and resistance to overfitting.

To ensure the model was trained on clean data, missing data was handled by **dropping rows with missing values** in the selected input features and the target variable. This was achieved using the `dropna()` method, which ensures that only rows with complete data were included in the training and testing process. Although this approach helps maintain data quality, it can also reduce the sample size, as rows containing any missing values were excluded.

The dataset was split into **training** and **test sets** using an 80-20 split, with the training set used to build the model and the test set reserved for performance evaluation. The parameter `random_state=42` was used to ensure **reproducibility**, making it possible to generate the same train-test split and model results consistently.

The model was trained using **100 decision trees** (n_estimators=100), which is a key feature of Random Forest models to reduce variance and improve predictive accuracy. After training, the model was evaluated using common metrics such as **Mean Squared Error (MSE)**, **Mean Absolute Error (MAE)**, and **R-squared** to gauge its performance. Additionally, feature importance was calculated to understand which environmental variables had the most significant impact on predicting air temperature.

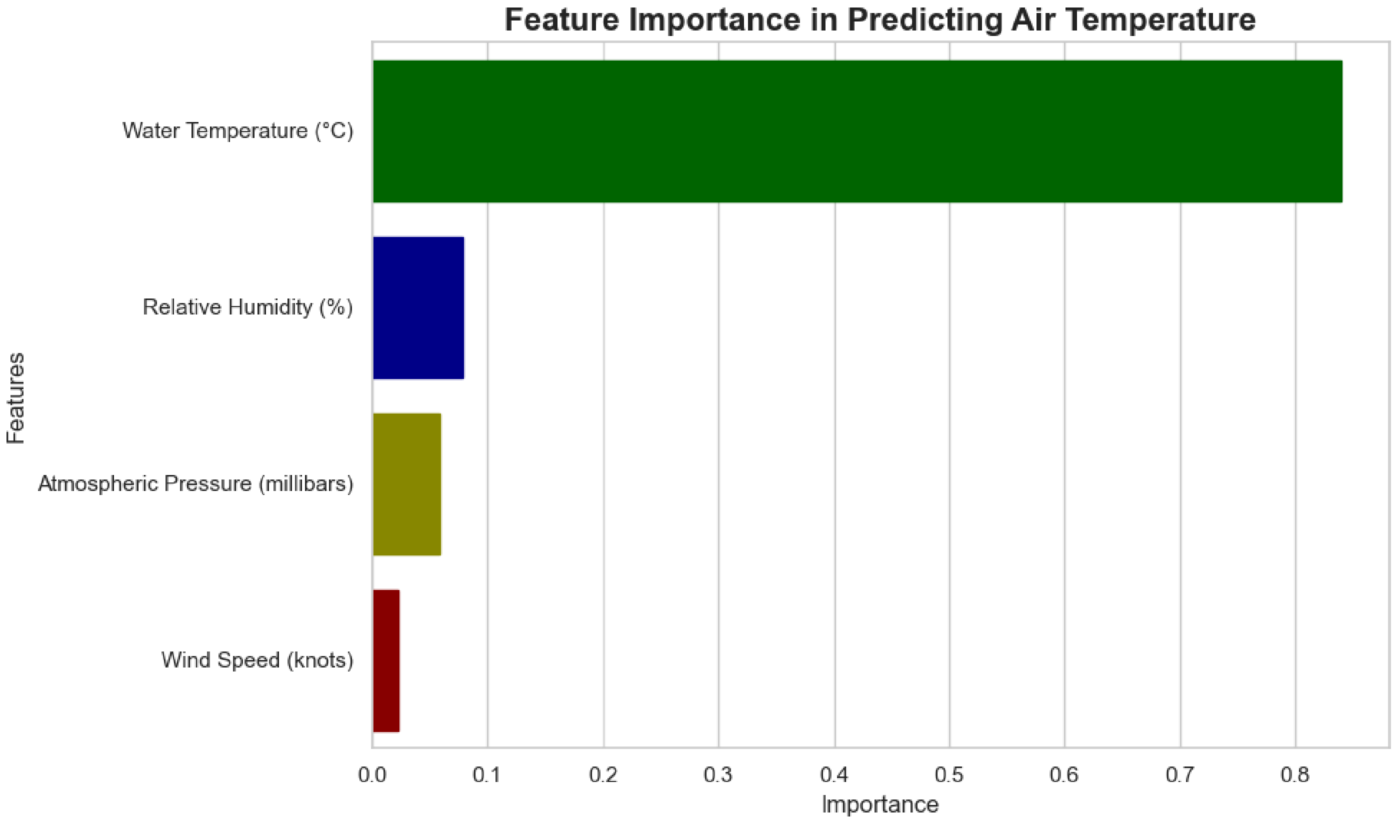
In [131…

```
import pandas as pd
import numpy as np
```

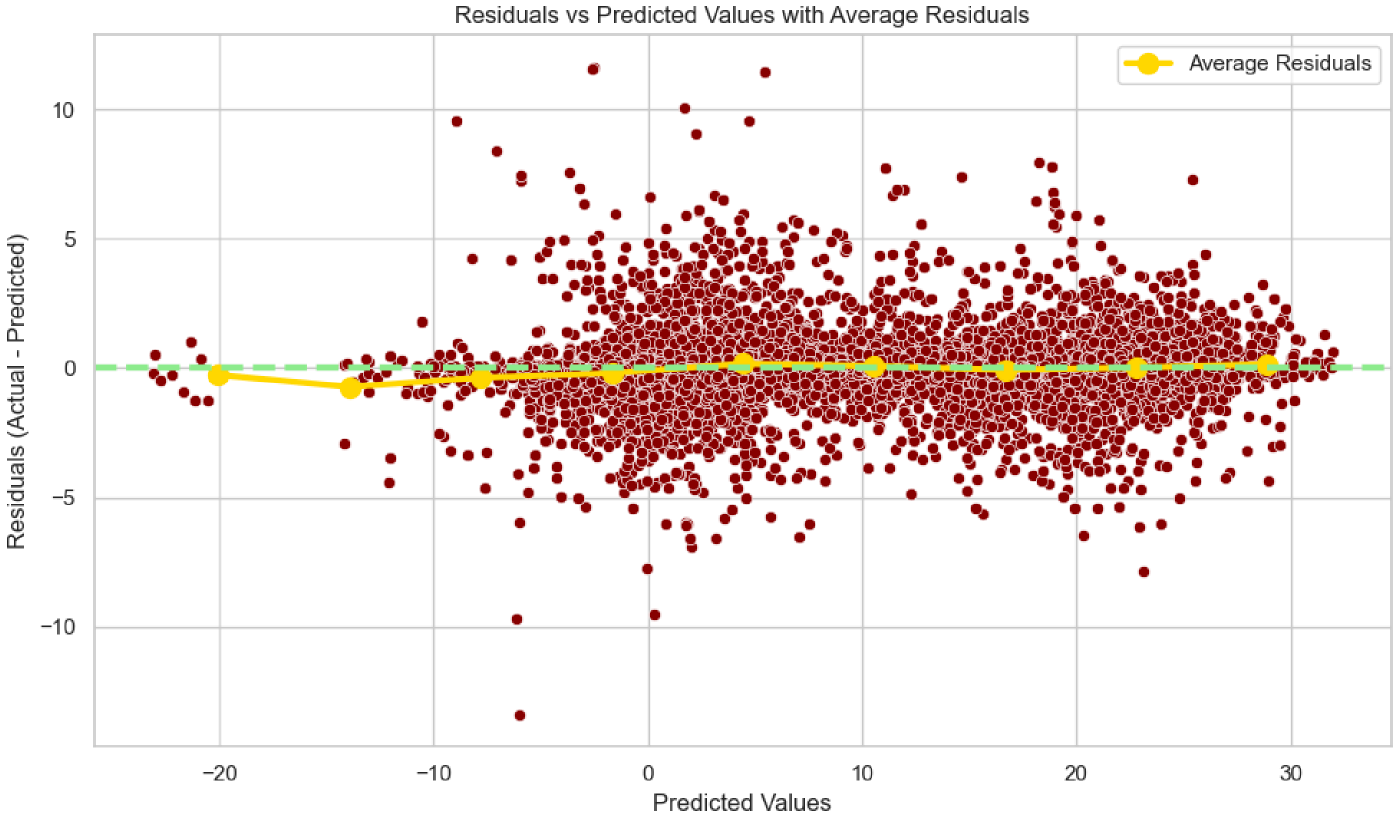
```python
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

# Define the feature descriptions mapping
feature_info = {
    'wtempc': 'Water Temperature (°C)',
    'relh_pct': 'Relative Humidity (%)',
    'windspeed_knots': 'Wind Speed (knots)',
    'mbars': 'Atmospheric Pressure (millibars)'
}

# Define the input features and target variable
features = ['wtempc', 'relh_pct', 'windspeed_knots', 'mbars']  # Input features
target = 'atempc'  # Target: air temperature

# Drop rows with missing values for these features
data = dev_df.dropna(subset=features + [target])

# Define X (features) and y (target)
X = data[features]  # Input features
y = data[target]  # Target variable: air temperature

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
rf_model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = rf_model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error (MSE): {mse}")
print(f"Mean Absolute Error (MAE): {mae}")
print(f"R-squared: {r2}")

# Calculate feature importance
feature_importances = rf_model.feature_importances_
features = X_train.columns

# Create a DataFrame for feature importance
feature_importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': feature_importances
})

# Replace feature names with their descriptions
feature_importance_df['Feature'] = feature_importance_df['Feature'].map(feature_info)
```

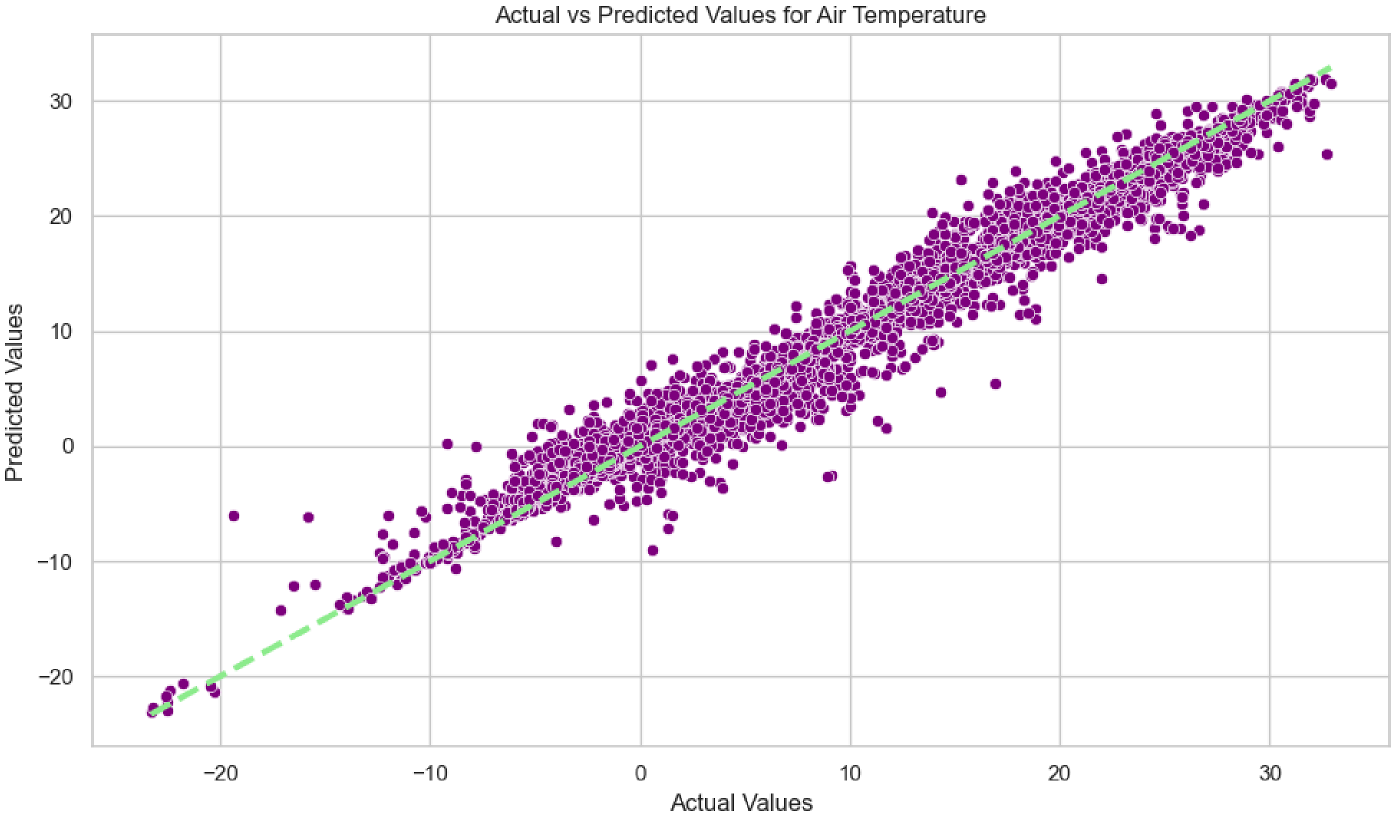
```
# Sort the features by importance
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

# Manually assign darker colors for each bar
colors = ['#006400', '#00008B', '#8B8B00','#8B0000']

# Plot feature importance
plt.figure(figsize=(10, 6))
bars = sns.barplot(x='Importance', y='Feature', data=feature_importance_df)

# Iterate over bars and assign colors manually
for i, bar in enumerate(bars.patches):
    bar.set_color(colors[i])

# Add titles and labels
plt.title('Feature Importance in Predicting Air Temperature', fontsize=16, fontweight='bold')
plt.xlabel('Importance', fontsize=12)
plt.ylabel('Features', fontsize=12)

# Show the plot
plt.tight_layout()
plt.show()

# Show the sorted feature importance DataFrame
print(feature_importance_df)

# Plot residuals vs predicted values
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_pred, y=y_test - y_pred, color='darkred')

# Calculate the average residuals in different bins
bins = np.linspace(min(y_pred), max(y_pred), 10)  # Create 10 bins
bin_centers = (bins[:-1] + bins[1:]) / 2  # Calculate bin centers
bin_indices = np.digitize(y_pred, bins)  # Get bin indices for each predicted value
average_residuals = [np.mean(y_test[bin_indices == i] - y_pred[bin_indices == i]) for i in range

# Plot the average residuals as a line plot
plt.plot(bin_centers, average_residuals, color='gold', linewidth=3, marker='o', markersize=10, la

# Overlay a horizontal line at zero
plt.axhline(0, color='lightgreen', linestyle='--', linewidth=3)
plt.title('Residuals vs Predicted Values with Average Residuals')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals (Actual - Predicted)')
plt.legend()
plt.tight_layout()
plt.show()

# Plot actual vs predicted values
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred, color='purple')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='lightgreen', linestyle='
plt.title('Actual vs Predicted Values for Air Temperature')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.tight_layout()
plt.show()
```

```
Mean Squared Error (MSE): 1.9151998341858187
Mean Absolute Error (MAE): 0.8279995023791433
R-squared: 0.9795919543959207
```



```
                           Feature  Importance
0           Water Temperature (°C)    0.839521
1             Relative Humidity (%)    0.079294
3  Atmospheric Pressure (millibars)    0.058727
2               Wind Speed (knots)    0.022458
```





Discussion

Feature Importance in Predicting Air Temperature

The bar plot displays the importance of various input features in predicting air temperature (`atempc`) using a random forest model. The **Water Temperature (°C)** is the most significant feature, contributing overwhelmingly to the predictions with an importance value of approximately 0.84. **Relative Humidity (%)** and **Atmospheric Pressure (millibars)** show moderate importance, while **Wind Speed (knots)** has the least impact. The strong contribution of water temperature suggests it is highly correlated with air temperature and should be the primary focus for predictions. The relatively low contributions of the other features indicate that additional features or more complex interactions may need to be explored to improve model performance.

Residuals vs. Predicted Values with Average Residuals

This plot shows the residuals (the difference between actual and predicted values) against the predicted values. The red points represent individual residuals, while the yellow line represents the average residuals for different ranges of predicted values. Ideally, residuals should be randomly scattered around zero with no clear pattern, indicating an unbiased model. In this plot, we observe some degree of heteroscedasticity (uneven spread), especially for higher predicted values, where the variance increases. This may suggest that the model struggles with predictions in this region. Nonetheless, the relatively flat yellow line shows that the model's errors, on average, are close to zero, indicating an overall balanced prediction.

Actual vs. Predicted Values for Air Temperature

This scatter plot compares actual air temperature values (on the x-axis) with the predicted values (on the y-axis). The green dashed line represents the perfect prediction scenario where predicted values exactly match the actual values. Most data points are close to this line, indicating that the model is performing well in predicting air temperature. However, there are still some deviations, particularly at the extremes, where the model either overpredicts or underpredicts. Despite these deviations, the model's performance is strong, with an R-squared value of around 0.98, which suggests that the model explains nearly all the variance in air temperature. However, further improvements could still be made by refining the model or adding more features.

Conclusion

In summary, this analysis provided valuable insights into the dataset, including the handling of missing values, exploring the relationships between features, and developing a predictive model using Random Forest regression. The model's performance was assessed using various evaluation metrics such as Mean Squared Error, Mean Absolute Error, and R-squared. Additionally, feature importance was analyzed to determine the most influential factors in predicting the target variable, and visualizations were used to further illustrate these relationships. While the model achieved good results, further improvements could be made through additional feature engineering, hyperparameter tuning, or exploring other algorithms to enhance prediction accuracy. This project also highlights the importance of reproducibility through environment management and proper documentation of the development tools.