



# Practical 06: Data Visualisation

Dr. Todd Jones

[t.r.jones@reading.ac.uk](mailto:t.r.jones@reading.ac.uk) (<mailto:t.r.jones@reading.ac.uk>)

**Department of Computer Science**

---

Follow the instructions to complete each of these tasks.

This set of exercises focuses on practicing implementation of data visualisation tools from `pandas` , `matplotlib` , and `seaborn` .

This work is not assessed but will help you gain practical experience for the coursework.

---

You will have accessed this practical notebook by downloading a `.zip` file containing the relevant datasets. The **relative path** to the data from this notebook should be:

`./data/<dataset_file_name>`

Please refer back to the **Data Visualisation lecture notebook** for internet references to various routines and packages.

Where **additional materials**, not covered in the lecture, are required, they are implemented for you or referenced below.

```
In [2]: # You will need these tools
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

---

## Histograms

Look at the `./data/faithful.csv` data, which gives a list of eruptions of a geyser in Yellowstone National Park in the USA. The `eruptions` variable gives the temporal length of the eruption, while the `waiting` variable gives the waiting time until the next eruption. One "source" on this very commonly explored dataset: <https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/faithful.html> (<https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/faithful.html>).

- Explore the data by looking at the **distribution** of the two variables.

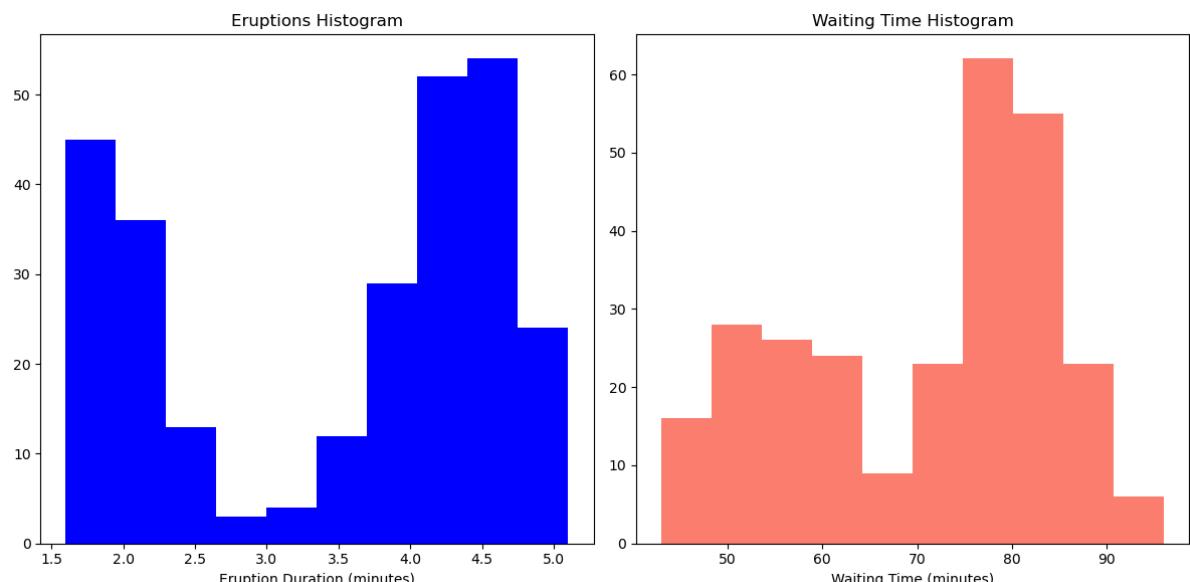
- Which kinds of plots look at distributions?
- Try out `plt.hist()` , `sns.histplot()` , and `sns.jointplot()` .
- How do they **vary together**?
- What do you notice about the distribution of the individual variables?
- Are they related?

In [5]: `# Simple read of the data  
data = pd.read_csv('./data/faithful.csv')  
data.head()`

Out[5]:

	eruptions	waiting
1	3.600	79
2	1.800	54
3	3.333	74
4	2.283	62
5	4.533	85

In [6]: `# Simple histogram with matplotlib  
plt.figure(figsize=(12, 6))  
plt.subplot(1, 2, 1)  
plt.hist(data['eruptions'], color='blue')  
plt.title('Eruptions Histogram')  
plt.xlabel('Eruption Duration (minutes)')  
  
plt.subplot(1, 2, 2)  
plt.hist(data['waiting'], color='salmon')  
plt.title('Waiting Time Histogram')  
plt.xlabel('Waiting Time (minutes)')  
  
plt.tight_layout()  
plt.show()`



```
In [21]: # Use sns.histplot
plt.figure(figsize=(12, 6))

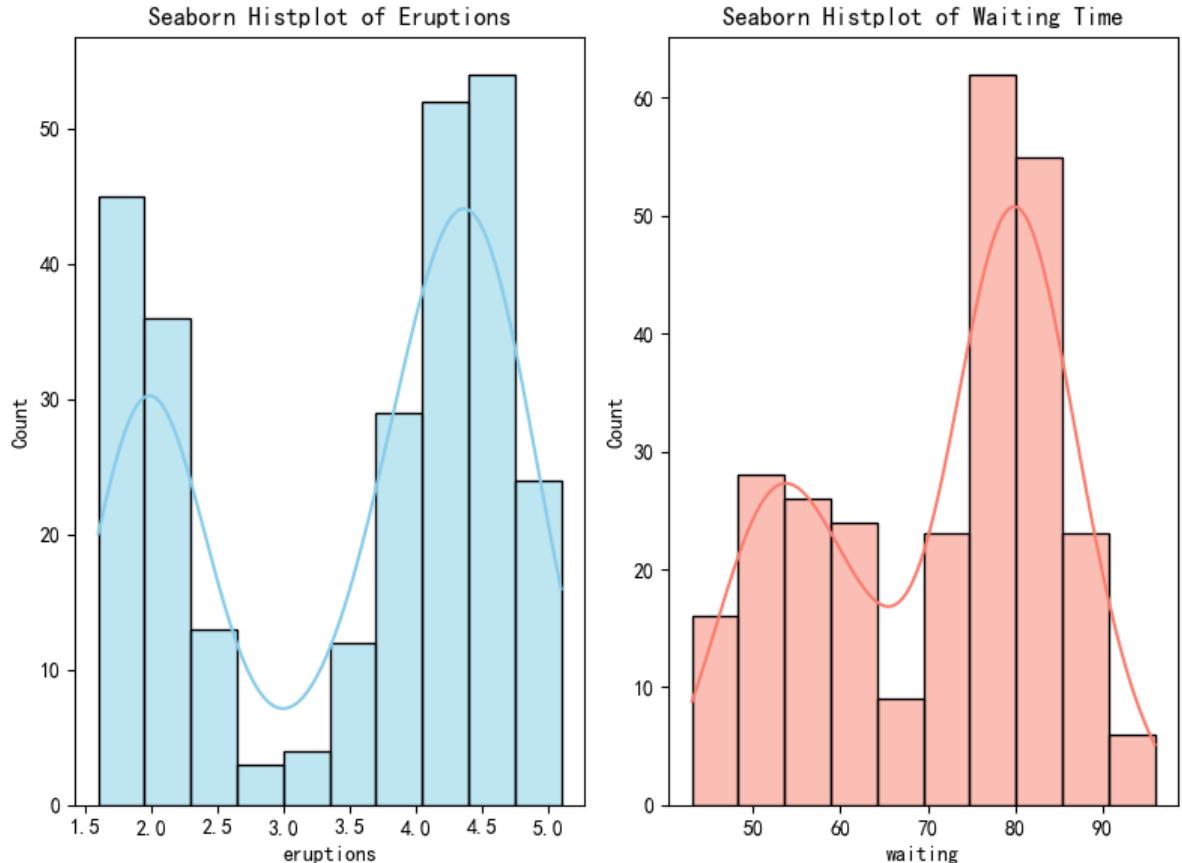
plt.subplot(1, 3, 1)
sns.histplot(data['eruptions'], kde=True, color='skyblue')
plt.title('Seaborn Histplot of Eruptions')

plt.subplot(1, 3, 2)
sns.histplot(data['waiting'], kde=True, color='salmon')
plt.title('Seaborn Histplot of Waiting Time')

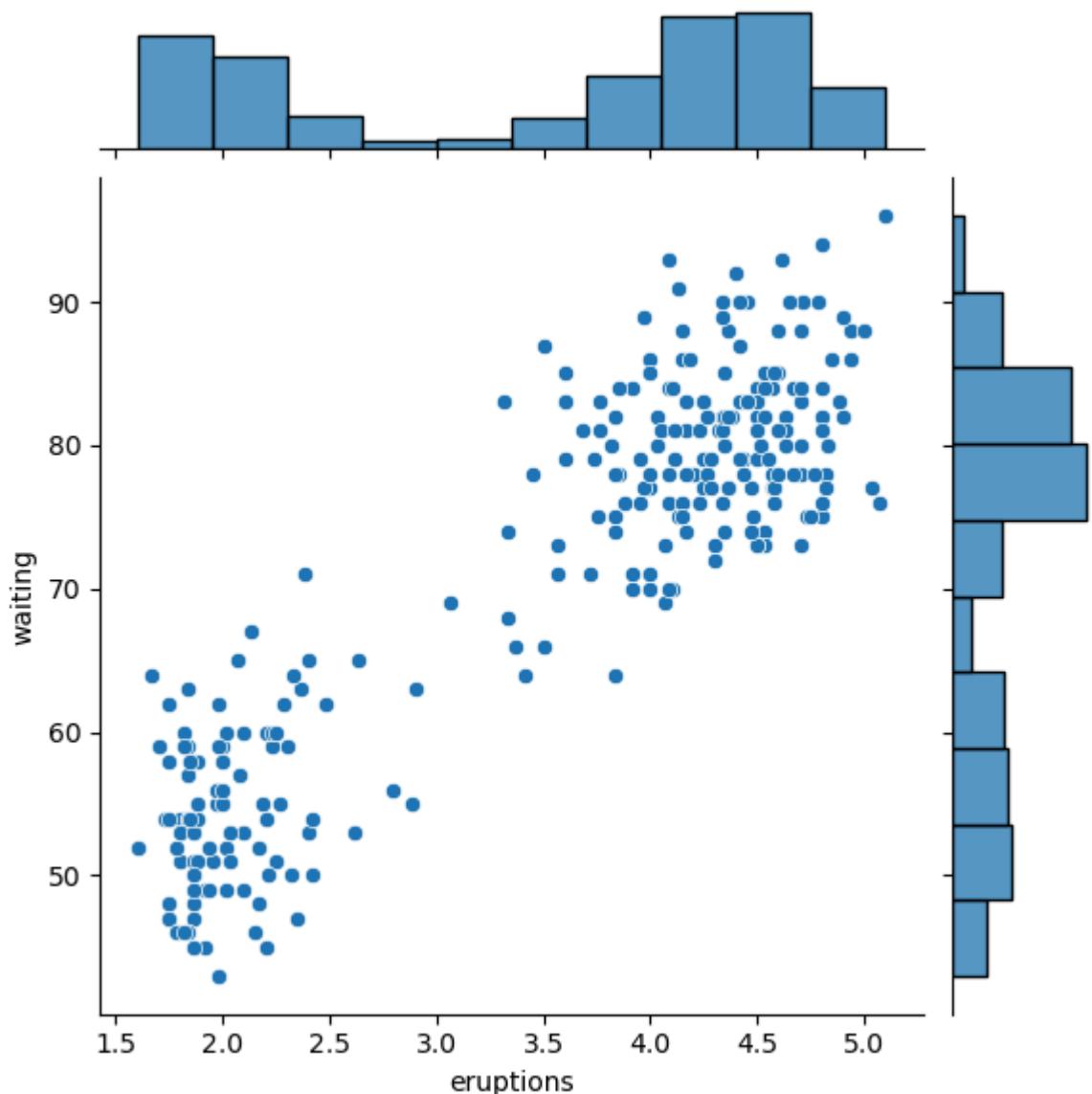
plt.tight_layout()
plt.show()
```

c:\Users\ybliu\anaconda3\envs\Mytorch\Lib\site-packages\seaborn\\_oldcore.py:1119: FutureWarning: use\_inf\_as\_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.

with pd.option\_context('mode.use\_inf\_as\_na', True):  
c:\Users\ybliu\anaconda3\envs\Mytorch\Lib\site-packages\seaborn\\_oldcore.py:1119: FutureWarning: use\_inf\_as\_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.  
with pd.option\_context('mode.use\_inf\_as\_na', True):



In [7]: # use jointplot to show the relationship between eruptions and waiting time  
 sns.jointplot(data=data, x='eruptions', y='waiting')  
 plt.show()



## Hong Kong Marine Water Quality

Hong Kong government open data: historical marine water quality

- Measurements of various **water quality** indicators in different monitoring stations.
- Data is split up into different zones.
- Data is measured at different times in different zones.

**Source:** <https://data.gov.hk/en-data/dataset/hk-epd-marineteam-marine-water-quality-historical-data-en> (<https://data.gov.hk/en-data/dataset/hk-epd-marineteam-marine-water-quality-historical-data-en>)

**Data:** ./data/marine-historical-2021-en.csv

**Description:** ./data/historical\_marine\_data\_dictionary\_en.pdf

## Read Data

Read the data and explore its contents.

```
In [8]: file = './data/marine-historical-2021-en.csv'
data = pd.read_csv(file)
```

```
In [9]: data
```

Out[9]:

	Water Control Zone	Station	Dates	Sample No	Depth	5-day Biochemical Oxygen Demand (mg/L)	Ammonia Nitrogen (mg/L)	Chlorophyll-a (µg/L)	Dissolved Oxygen (%saturation)
0	Deep Bay	DM1	2021-02-27	1	Surface Water	1.4	0.460	4.1	7
1	Deep Bay	DM1	2021-03-24	1	Surface Water	12	0.270	14	7
2	Deep Bay	DM1	2021-04-16	1	Surface Water	2	0.260	15	6
3	Deep Bay	DM1	2021-05-17	1	Surface Water	1.4	0.220	14	5
4	Deep Bay	DM1	2021-06-16	1	Surface Water	2.1	0.410	1.7	5
...	...	...	...	...	...	...	...	...	...
2703	Western Buffer	WT3	2021-09-06	1	Middle Water	1.1	0.058	10	6
2704	Western Buffer	WT3	2021-09-06	1	Bottom Water	0.5	0.050	5.7	5
2705	Western Buffer	WT3	2021-11-01	1	Surface Water	0.4	0.045	1.1	8
2706	Western Buffer	WT3	2021-11-01	1	Middle Water	0.4	0.046	0.9	6
2707	Western Buffer	WT3	2021-11-01	1	Bottom Water	0.4	0.048	1.4	8

2708 rows × 29 columns



In [10]: `data.columns`

Out[10]: Index(['Water Control Zone', 'Station', 'Dates', 'Sample No', 'Depth', '5-day Biochemical Oxygen Demand (mg/L)', 'Ammonia Nitrogen (mg/L)', 'Chlorophyll-a (\u00b5g/L)', 'Dissolved Oxygen (%saturation)', 'Dissolved Oxygen (mg/L)', 'E. coli (cfu/100mL)', 'Faecal Coliforms (cfu/100mL)', 'Nitrate Nitrogen (mg/L)', 'Nitrite Nitrogen (mg/L)', 'Orthophosphate Phosphorus (mg/L)', 'pH', 'Phaeo-pigments (\u00b5g/L)', 'Salinity (psu)', 'Secchi Disc Depth (M)', 'Silica (mg/L)', 'Suspended Solids (mg/L)', 'Temperature (\u00b0 C)', 'Total Inorganic Nitrogen (mg/L)', 'Total Kjeldahl Nitrogen (mg/L)', 'Total Nitrogen (mg/L)', 'Total Phosphorus (mg/L)', 'Turbidity (NTU)', 'Unionised Ammonia (mg/L)', 'Volatile Suspended Solids (mg/L')],  
dtype='object')

In [11]: `data.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2708 entries, 0 to 2707
Data columns (total 29 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   Water Control Zone    2708 non-null   object  
 1   Station            2708 non-null   object  
 2   Dates              2708 non-null   object  
 3   Sample No          2708 non-null   int64   
 4   Depth              2708 non-null   object  
 5   5-day Biochemical Oxygen Demand (mg/L) 2708 non-null   object  
 6   Ammonia Nitrogen (mg/L)      2708 non-null   float64 
 7   Chlorophyll-a (\u00b5g/L)     2708 non-null   object  
 8   Dissolved Oxygen (%saturation) 2708 non-null   int64   
 9   Dissolved Oxygen (mg/L)      2708 non-null   float64 
 10  E. coli (cfu/100mL)        2708 non-null   object  
 11  Faecal Coliforms (cfu/100mL) 2708 non-null   object  
 12  Nitrate Nitrogen (mg/L)     2708 non-null   object  
 13  Nitrite Nitrogen (mg/L)     2708 non-null   object  
 14  Orthophosphate Phosphorus (mg/L) 2708 non-null   object  
 15  pH                  2708 non-null   float64 
 16  Phaeo-pigments (\u00b5g/L)    2708 non-null   object  
 17  Salinity (psu)           2708 non-null   float64 
 18  Secchi Disc Depth (M)    2708 non-null   float64 
 19  Silica (mg/L)           2708 non-null   object  
 20  Suspended Solids (mg/L)   2708 non-null   object  
 21  Temperature (\u00b0 C)       2708 non-null   float64 
 22  Total Inorganic Nitrogen (mg/L) 2708 non-null   float64 
 23  Total Kjeldahl Nitrogen (mg/L) 1214 non-null   object  
 24  Total Nitrogen (mg/L)      1214 non-null   float64 
 25  Total Phosphorus (mg/L)    1214 non-null   object  
 26  Turbidity (NTU)           2705 non-null   float64 
 27  Unionised Ammonia (mg/L)  2708 non-null   object  
 28  Volatile Suspended Solids (mg/L) 2708 non-null   object  
dtypes: float64(9), int64(2), object(18)
memory usage: 613.7+ KB
```

In [12]: `data.describe()`

Out[12]:

	Sample No	Ammonia Nitrogen (mg/L)	Dissolved Oxygen (%saturation)	Dissolved Oxygen (mg/L)	pH	Salinity (psu)	Secchi Disc Depth (M)
<b>count</b>	2708.0	2708.000000	2708.000000	2708.000000	2708.000000	2708.000000	2708.000000
<b>mean</b>	1.0	0.078558	72.679099	5.054247	7.612075	31.603028	2.93017
<b>std</b>	0.0	0.091631	13.517548	0.985105	0.245578	3.194204	1.35502
<b>min</b>	1.0	0.010000	16.000000	1.100000	6.100000	7.100000	0.70000
<b>25%</b>	1.0	0.030000	65.000000	4.500000	7.500000	31.200000	2.10000
<b>50%</b>	1.0	0.049000	71.000000	5.000000	7.600000	32.600000	2.60000
<b>75%</b>	1.0	0.099000	82.000000	5.700000	7.800000	33.400000	3.32500
<b>max</b>	1.0	1.300000	157.000000	10.200000	8.300000	34.500000	11.30000

## Re-Read Data

Read in the data again, this time specifying that the second column be used as the DataFrame index and ask `read_csv` to `parse_dates`.

- Use the `index_col` parameter to set which input column should be used to represent the index of the DataFrame.

In [13]: `data = pd.read_csv(file, index_col=2, parse_dates=True)`

In [14]: data

Out[14]:

	Water Control Zone	Station	Sample No	Depth	5-day Biochemical Oxygen Demand (mg/L)	Ammonia Nitrogen (mg/L)	Chlorophyll-a (µg/L)	Dissolved Oxygen (%saturation)	Dis C
<b>Dates</b>									
2021-02-27	Deep Bay	DM1	1	Surface Water	1.4	0.460	4.1	72	
2021-03-24	Deep Bay	DM1	1	Surface Water	12	0.270	14	70	
2021-04-16	Deep Bay	DM1	1	Surface Water	2	0.260	15	65	
2021-05-17	Deep Bay	DM1	1	Surface Water	1.4	0.220	14	57	
2021-06-16	Deep Bay	DM1	1	Surface Water	2.1	0.410	1.7	56	
...	...	...	...	...	...	...	...	...	...
2021-09-06	Western Buffer	WT3	1	Middle Water	1.1	0.058	10	62	
2021-09-06	Western Buffer	WT3	1	Bottom Water	0.5	0.050	5.7	59	
2021-11-01	Western Buffer	WT3	1	Surface Water	0.4	0.045	1.1	88	
2021-11-01	Western Buffer	WT3	1	Middle Water	0.4	0.046	0.9	67	
2021-11-01	Western Buffer	WT3	1	Bottom Water	0.4	0.048	1.4	82	

2708 rows × 28 columns



## Clean Data

Call the `clean_data` function on your `DataFrame` to replace the small value strings with zeros.

In [15]: # Convert each column to a float

```
def clean_data(indata):  
    '''
```

This function will modify the indata DataFrame inplace.

The HK data has several features that record small numerical values as strings that start with "<".

This code converts data from column 4 onward to floats, replacing the small values with zero.

```
'''
```

```
for col in indata.columns[4]:
```

```
    if indata[col].dtype == 'object':
```

```
        try:
```

```
            indata[col] = indata[col].astype(float)
```

```
        except Exception as e:
```

```
            print(col)
```

```
            print('\t', e)
```

```
        try:
```

```
            indata[col] = indata[col].where(~indata[col].astype(str).str.starts
```

```
            indata[col] = indata[col].astype(float)
```

```
        except Exception as e:
```

```
            print('ERROR: Our modification did not work.')
```

```
            print(e)
```

```
    print()
```

```
indata.info()
```

```
In [16]: # Calling function to clean the data  
clean_data(data)
```

5-day Biochemical Oxygen Demand (mg/L)  
could not convert string to float: '<0.1'  
Chlorophyll-a ( $\mu$ g/L)  
could not convert string to float: '<0.2'  
E. coli (cfu/100mL)  
could not convert string to float: '<1'  
Faecal Coliforms (cfu/100mL)  
could not convert string to float: '<1'  
Nitrate Nitrogen (mg/L)  
could not convert string to float: '<0.002'  
Nitrite Nitrogen (mg/L)  
could not convert string to float: '<0.002'  
Orthophosphate Phosphorus (mg/L)  
could not convert string to float: '<0.002'  
Phaeo-pigments ( $\mu$ g/L)  
could not convert string to float: '<0.2'  
Silica (mg/L)  
could not convert string to float: '<0.05'  
Suspended Solids (mg/L)  
could not convert string to float: '<0.5'  
Total Kjeldahl Nitrogen (mg/L)  
could not convert string to float: '<0.05'  
Total Phosphorus (mg/L)  
could not convert string to float: '<0.02'  
Unionised Ammonia (mg/L)  
could not convert string to float: '<0.001'  
Volatile Suspended Solids (mg/L)  
could not convert string to float: '<0.5'

<class 'pandas.core.frame.DataFrame'>  
DatetimeIndex: 2708 entries, 2021-02-27 to 2021-11-01  
Data columns (total 28 columns):

#	Column	Non-Null Count	Dtype
0	Water Control Zone	2708	object
1	Station	2708	object
2	Sample No	2708	int64
3	Depth	2708	object
4	5-day Biochemical Oxygen Demand (mg/L)	2708	float64
5	Ammonia Nitrogen (mg/L)	2708	float64
6	Chlorophyll-a ( $\mu$ g/L)	2708	float64
7	Dissolved Oxygen (%saturation)	2708	int64
8	Dissolved Oxygen (mg/L)	2708	float64
9	E. coli (cfu/100mL)	2708	float64
10	Faecal Coliforms (cfu/100mL)	2708	float64
11	Nitrate Nitrogen (mg/L)	2708	float64
12	Nitrite Nitrogen (mg/L)	2708	float64
13	Orthophosphate Phosphorus (mg/L)	2708	float64
14	pH	2708	float64
15	Phaeo-pigments ( $\mu$ g/L)	2708	float64
16	Salinity (psu)	2708	float64
17	Secchi Disc Depth (M)	2708	float64
18	Silica (mg/L)	2708	float64
19	Suspended Solids (mg/L)	2708	float64
20	Temperature (° C)	2708	float64
21	Total Inorganic Nitrogen (mg/L)	2708	float64
22	Total Kjeldahl Nitrogen (mg/L)	1214	non-null
23	Total Nitrogen (mg/L)	1214	non-null
24	Total Phosphorus (mg/L)	1214	non-null
25	Turbidity (NTU)	2705	non-null
26	Unionised Ammonia (mg/L)	2708	float64

```
27  Volatile Suspended Solids (mg/L)      2708 non-null    float64
dtypes: float64(23), int64(2), object(3)
memory usage: 613.5+ KB
```

## pandas Line Plots

Use `.plot()` from pandas to make a simple line plot showing a time series of 'Dissolved Oxygen (mg/L)' for the **entire dataset** (all locations).

**Some of this plot will look as though there are very linear periods (instead of highly variable). What are these linear features?**

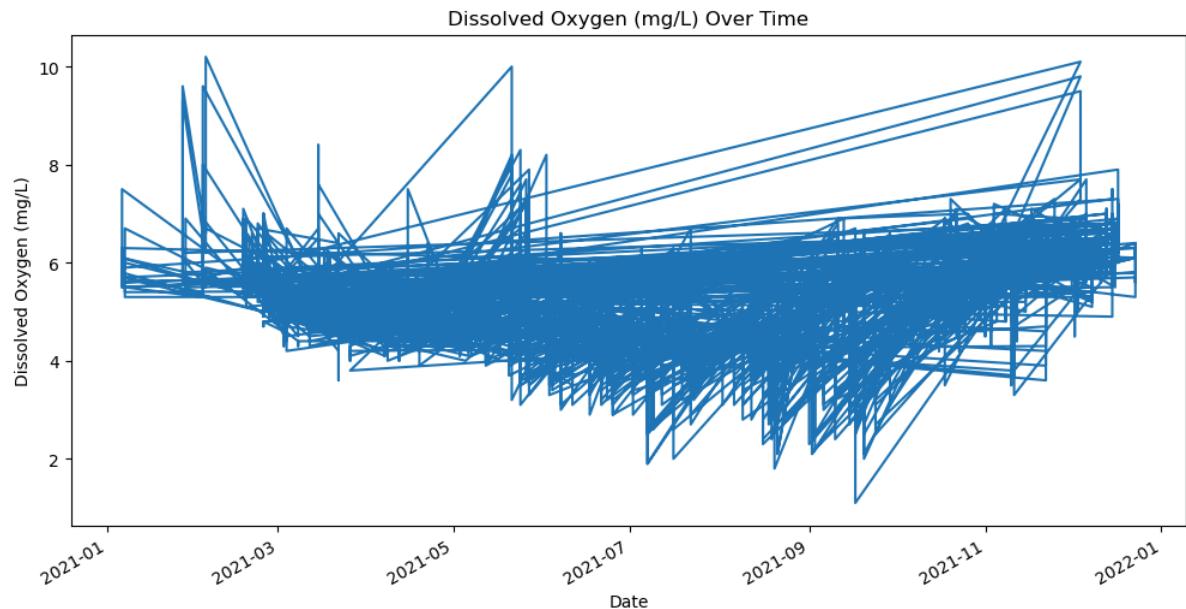
- Were any plot labels automatically generated?
- How can you add any missing information?

In [17]:

```
# the date is the index now
data['Dissolved Oxygen (mg/L)'].plot(figsize=(12, 6))

plt.title('Dissolved Oxygen (mg/L) Over Time')
plt.xlabel('Date')
plt.ylabel('Dissolved Oxygen (mg/L)')
```

Out[17]:

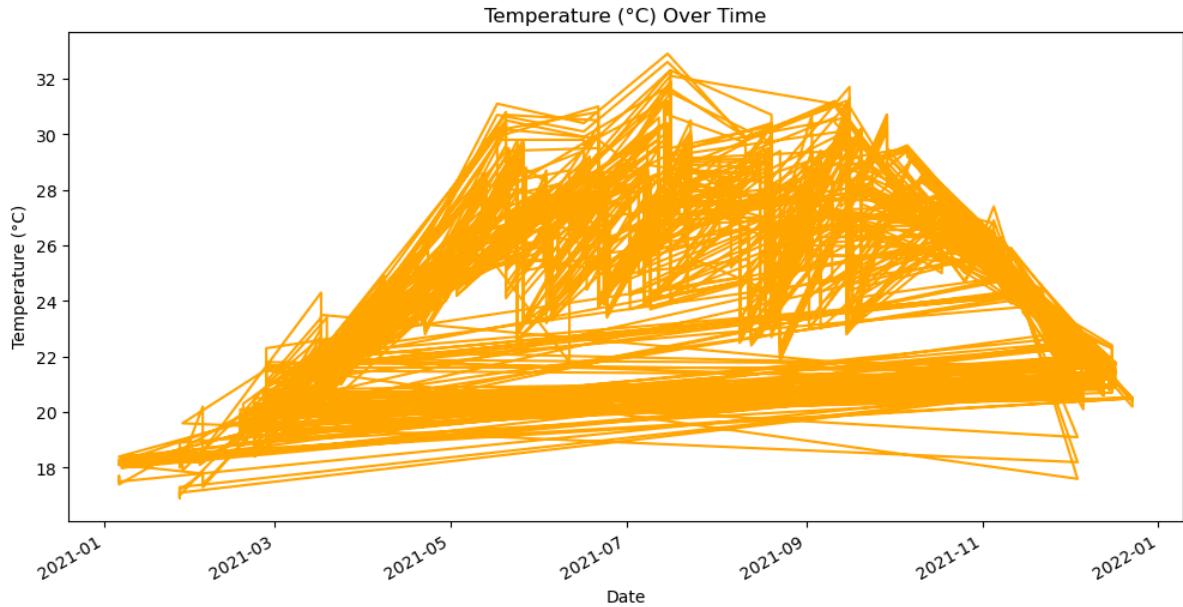


Create a similar plot for Temperature ( $^{\circ}$  C) .

**Does the result make logical sense? That is, do warmer and colder temperatures occur at appropriate times of the year?**

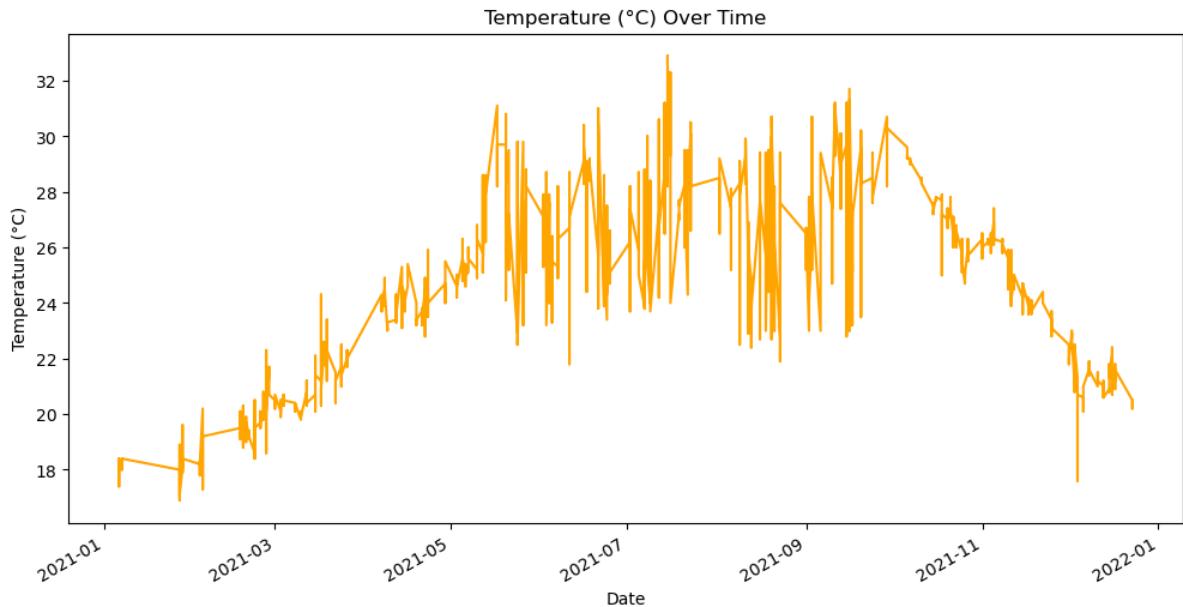
```
In [18]: data['Temperature (° C)'].plot(figsize=(12, 6), color='orange')
plt.title('Temperature (° C) Over Time')
plt.xlabel('Date')
plt.ylabel('Temperature (° C)')
```

Out[18]: Text(0, 0.5, 'Temperature (° C)')



```
In [19]: data['Temperature (° C)'].sort_index().plot(figsize=(12, 6), color='orange')
plt.title('Temperature (° C) Over Time')
plt.xlabel('Date')
plt.ylabel('Temperature (° C)')
```

Out[19]: Text(0, 0.5, 'Temperature (° C)')



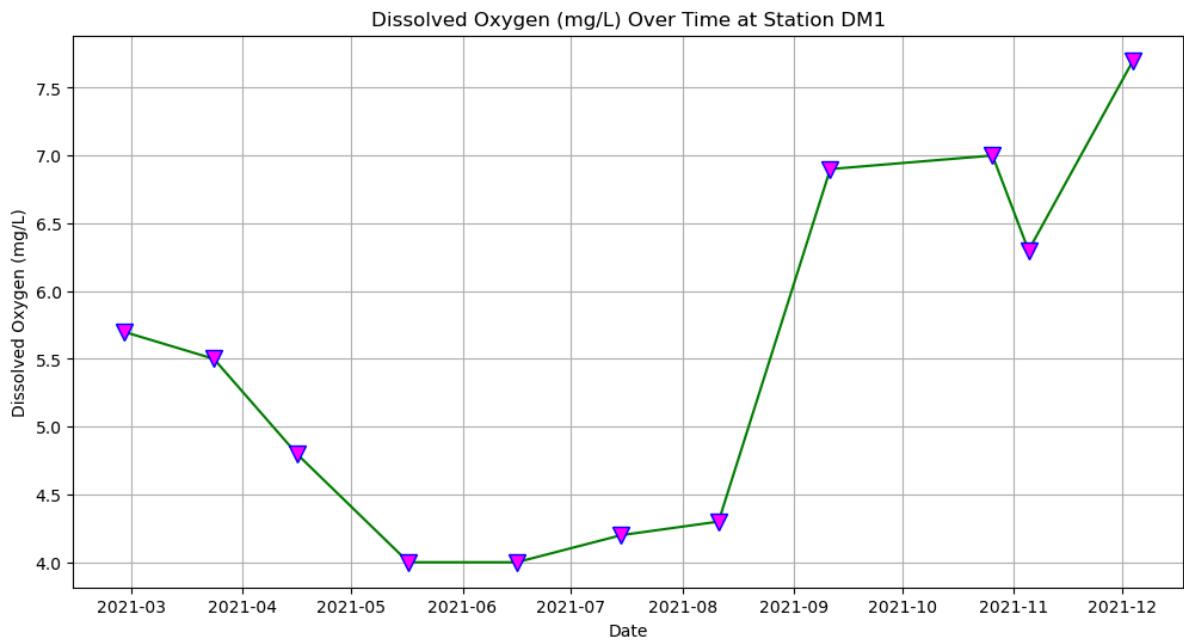
## Customising Plot Elements

Plot a time series of 'Dissolved Oxygen (mg/L)' for the DM1 **Station** using `pandas`. Ensure that the y-axis is labelled appropriately.

- You will need to rely on `plt.ylabel()`.
- See if you can also make each individual data point on the line to display as a downward-pointing triangle.
  - [`https://matplotlib.org/stable/api/markers\_api.html`](https://matplotlib.org/stable/api/markers_api.html)  
`(https://matplotlib.org/stable/api/markers_api.html)`
- Can you make the line **green**?

```
In [20]: dm1_data = data[data['Station'] == 'DM1']
fig, ax = plt.subplots(figsize=(12, 6))
ax.plot(dm1_data.index, dm1_data['Dissolved Oxygen (mg/L)'], marker='v', color='green',
        plt.title('Dissolved Oxygen (mg/L) Over Time at Station DM1')
        plt.ylabel('Dissolved Oxygen (mg/L)')
        plt.xlabel('Date')

plt.grid(True)
plt.show()
```



## seaborn Violin Plots

Use the **seaborn** `catplot` utility to create a set of **violin plots** describing the distribution of pH in each Water Control Zone .

Place the **categorical** variable along the x-axis and the **numerical** variable on the y-axis.

Investigate the `rotation` parameter associated with `set_xticklabels` to ensure that each label can easily be read. You might also find the `ha` parameter to provide more aesthetically pleasing results.

To modify the ticks, save the resulting plot object and use `set_` methods on that object.

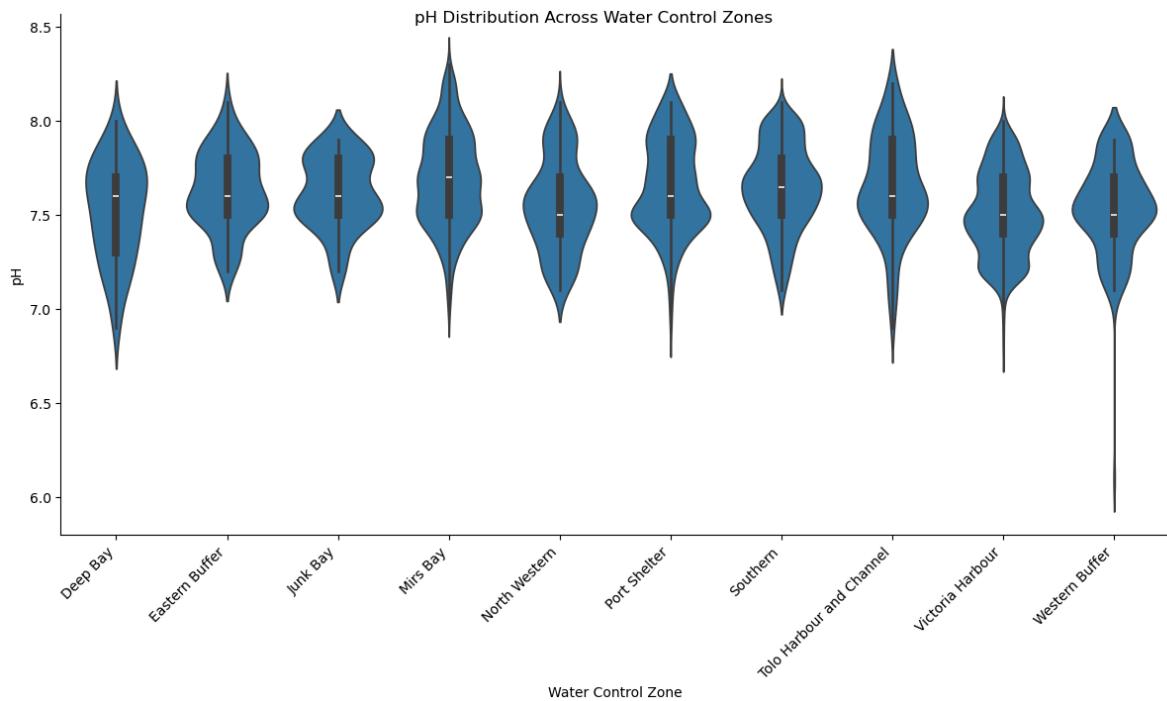
See: [https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.axes.Axes.set\\_xticklabels.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xticklabels.html)  
[https://matplotlib.org/stable/\\_as\\_gen/matplotlib.axes.Axes.set\\_xticklabels.html](https://matplotlib.org/stable/_as_gen/matplotlib.axes.Axes.set_xticklabels.html)

```
In [21]: g = sns.catplot(
    data=data,
    kind='violin',
    x='Water Control Zone',
    y='pH',
    height=6,
    aspect=2
)

# 旋转 x 轴标签
g.set_xticklabels(rotation=45, ha='right')

# 设置标题
g.fig.suptitle('pH Distribution Across Water Control Zones')

plt.show()
```



## seaborn Bar Plots

Use the `seaborn catplot` utility to create a group of **bar plots**. You will create an array of subplots (`catplot` can do this automatically) where each subplot (i.e., the `col` parameter), corresponding to individual Water Control Zone locations, displays the measurement of 'E. coli (cfu/100mL)' in each month. The plotting routine will consider all entries with the same month name in the same water control zone. The bars will represent the mean measurement for each of these entries, and variability will be represented with a line through the end of the bar.

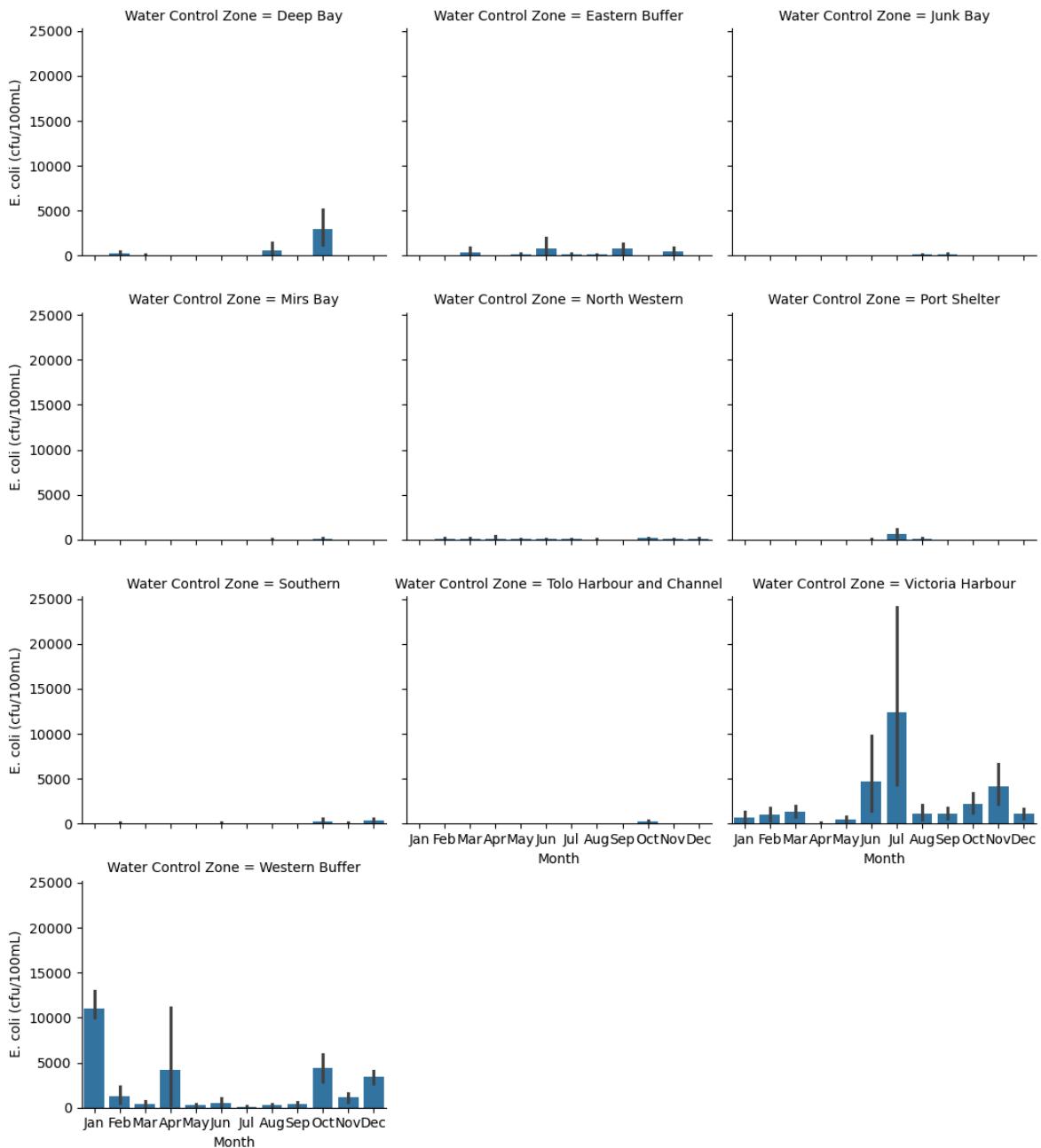
To organise the data by month, we will need to create a new 'Month' column in the DataFrame . To do this, we will convert the time values in the index to **string abbreviations** for calendar months by using the `strftime` function with a suitable **format code**. See:  
<https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior>  
(<https://docs.python.org/3/library/datetime.html#strftime-strptime-behavior>)

Appropriate representations will look like:

```
[ 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov',  
  'Dec' ]
```

Try to display these neatly by using the `col_wrap` parameter and with the months displayed in time order from Jan-Dec by using the `order` parameter.

```
In [22]: data['Month'] = data.index.strftime('%b')
month_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
               'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec']
g = sns.catplot(data=data, x='Month', y='E. coli (cfu/100mL)', col='Water Control Zone')
```



## Comparing Zones

### Exploring the Data

We would like to **compare** how certain measurements change in time in different zones. Let's investigate how this might look for a specific measurement in a specific zone.

Create a **timeseries** plot of 'Salinity (psu)' for 'Victoria Harbour'. In this plot, **do not display the line** between the data points (e.g., `linestyle=''`). Instead, only show the data points as **circular markers**.

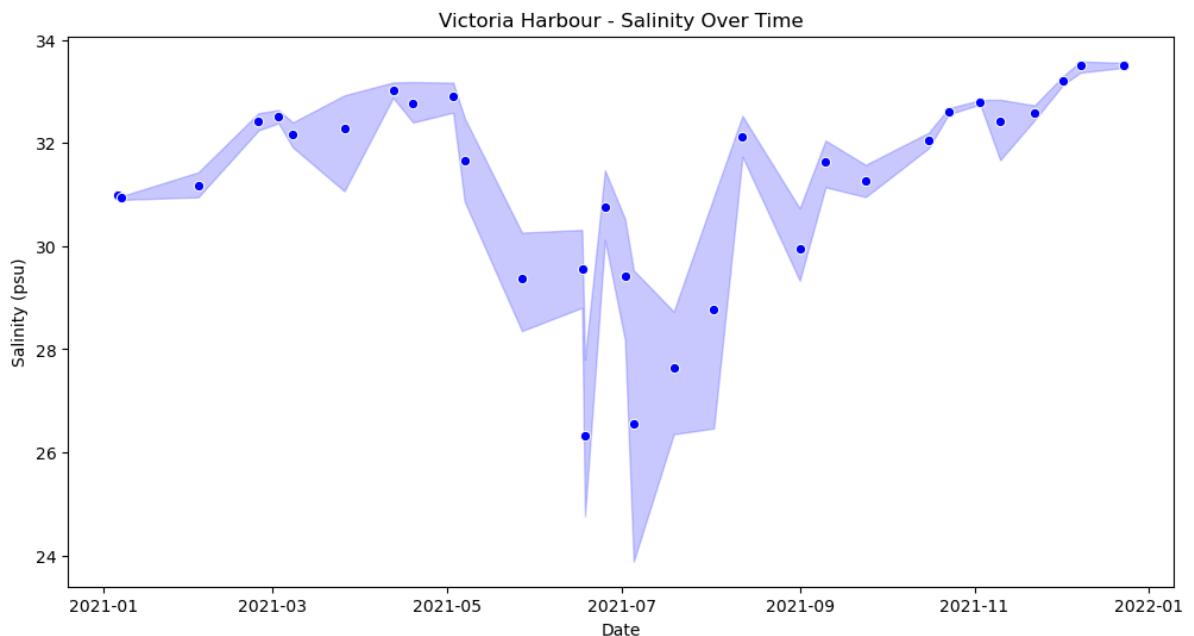
In [23]:

```
vh_data = data[data['Water Control Zone'] == 'Victoria Harbour']
plt.figure(figsize=(12, 6))
sns.lineplot(x=vh_data.index, y=vh_data['Salinity (psu)'], color='blue', marker='o', linestyle='')

plt.xlabel('Date')
plt.ylabel('Salinity (psu)')
plt.title('Victoria Harbour - Salinity Over Time')
```

Out[23]:

Text(0.5, 1.0, 'Victoria Harbour - Salinity Over Time')



In the previous plot, it appears that there are many days without measurements and several days for which there are multiple measurements.

Confirm this suspicion by inspecting the `index` entries of the **entire** data set or a subset, such as specifically Victoria Harbour, or specific stations in Victoria Harbour.

- How many entries are present in the dataset?
- How many **unique** entries are present in the `index` ?
- How many **duplicate** entries are present in the `index` ?

In [24]:

```
# For the entire dataset
total_entries = len(data)
unique_entries = data.index.unique()
duplicates = data.index.duplicated().sum()
print(f'Total entries: {total_entries}')
print(f'Unique entries: {len(unique_entries)}')
print(f'Duplicates: {duplicates}'')
```

Total entries: 2708  
 Unique entries: 142  
 Duplicates: 2566

```
In [25]: # For Victoria Harbour
vh_data = data[data['Water Control Zone'] == 'Victoria Harbour']
total_entries = len(vh_data)
unique_entries = vh_data.index.unique()
duplicates = vh_data.index.duplicated().sum()
print(f'Total entries: {total_entries}')
print(f'Unique entries: {len(unique_entries)}')
print(f'Duplicates: {duplicates}'')
```

Total entries: 438  
 Unique entries: 31  
 Duplicates: 407

```
In [26]: # Report the number of duplicated dates for each station in Victoria Harbour
duplicate_dates_by_station = vh_data.groupby('Station').apply(lambda x: x.index.duplicated())
print(duplicate_dates_by_station)
```

Station	
VM1	22
VM12	22
VM14	24
VM15	24
VM2	22
VM4	24
VM5	24
VM6	22
VM7	22
VM8	22
VT10	6
VT11	12
VT12	6
VT2	6
VT3	6
VT4	12
VT8	6
	dtype: int32

C:\Users\liuyuhao\AppData\Local\Temp\ipykernel\_10764\2813116721.py:2: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a future version of pandas the grouping columns will be excluded from the operation. Either pass `include\_groups=False` to exclude the groupings or explicitly select the grouping columns after groupby to silence this warning.

```
duplicate_dates_by_station = vh_data.groupby('Station').apply(lambda x: x.index.duplicated().sum())
```

## Reducing the Data

Now that we see that some days and stations have multiple measurements, let's **reduce** the total number of data points by averaging out the representation of multiple measurements and multiple stations.

**Group** the data by 'Water Control Zone' and 'Dates'. Then, **average** the grouped data to obtain average measurements for each zone on each date.

**Store** the result in a new object.

Display the resulting DataFrame

```
In [29]: # numerical_cols = data.select_dtypes(include=['float64', 'int64']).columns
# non_numerical_cols = data.select_dtypes(exclude=['float64', 'int64']).columns

# df = data.groupby(['Water Control Zone', 'Dates']).agg(
#     **{col: 'mean' for col in numerical_cols}, # average of numerical columns
#     **{col: 'first' for col in non_numerical_cols}
# )

# print(df.head(10))
df = data.groupby(['Water Control Zone', 'Dates']).mean(numeric_only=True)
df
```

Out[29]:

		Sample No	5-day Biochemical Oxygen Demand (mg/L)	Ammonia Nitrogen (mg/L)	Chlorophyll-a (µg/L)	Dissolved Oxygen (%saturation)	Dissolved Oxygen (mg/L)	E. coli (cfu/100mL)
Water Control Zone	Dates							
	2021-01-28	1.0	2.100000	0.100000	15.000000	71.000000	5.600000	4.000000
	2021-02-27	1.0	1.250000	0.181250	3.762500	70.250000	5.400000	280.750000
Deep Bay	2021-03-24	1.0	2.775000	0.165000	12.325000	69.125000	5.175000	47.125000
	2021-04-16	1.0	4.212500	0.142875	37.787500	71.250000	5.050000	11.000000
	2021-05-17	1.0	0.887500	0.055875	9.237500	66.500000	4.612500	9.000000
	...	...	...	...	...	...	...	...
	2021-09-06	1.0	0.766667	0.063667	8.133333	61.833333	4.116667	159.166667
	2021-10-11	1.0	0.325000	0.137750	0.608333	79.500000	5.200000	4371.666667
Western Buffer	2021-11-01	1.0	0.366667	0.054000	1.183333	76.333333	5.100000	207.666667
	2021-11-08	1.0	1.166667	0.088917	3.516667	87.083333	5.850000	1611.666667
	2021-12-08	1.0	0.500000	0.128667	1.300000	79.250000	5.741667	3383.333333

183 rows × 25 columns



## Reformatting the Data

In the previous result, you will see that the index now has two components: 'Water Control Zone' and 'Dates'. If you inspect the index, you will see that it is a MultiIndex object with a hierarchical structure.

You can use the `.unstack()` method on the DataFrame to **pivot** a level of the MultiIndex so that it will become the **inner** component of the column structure. -

[\(https://pandas.pydata.org/pandas-docs/version/2.2.2/reference/api/pandas.DataFrame.unstack.html\)](https://pandas.pydata.org/pandas-docs/version/2.2.2/reference/api/pandas.DataFrame.unstack.html)

Let's focus on the 'Suspended Solids (mg/L)' variable. Select this variable from the grouped and averaged data and **unstack** the Water Control Zone. By default, `unstack()` will operate on the *last* level, so you will need to specify the correct level by name or number.

**Display** the unstacked result. You should see zone names as columns and dates as the indices.

In [30]: `df.index`

```
Out[30]: MultiIndex([( 'Deep Bay', '2021-01-28'),
                      ( 'Deep Bay', '2021-02-27'),
                      ( 'Deep Bay', '2021-03-24'),
                      ( 'Deep Bay', '2021-04-16'),
                      ( 'Deep Bay', '2021-05-17'),
                      ( 'Deep Bay', '2021-06-16'),
                      ( 'Deep Bay', '2021-07-15'),
                      ( 'Deep Bay', '2021-08-11'),
                      ( 'Deep Bay', '2021-09-11'),
                      ( 'Deep Bay', '2021-10-15'),
                      ...
                      ('Western Buffer', '2021-06-11'),
                      ('Western Buffer', '2021-07-07'),
                      ('Western Buffer', '2021-07-12'),
                      ('Western Buffer', '2021-08-21'),
                      ('Western Buffer', '2021-09-02'),
                      ('Western Buffer', '2021-09-06'),
                      ('Western Buffer', '2021-10-11'),
                      ('Western Buffer', '2021-11-01'),
                      ('Western Buffer', '2021-11-08'),
                      ('Western Buffer', '2021-12-08')],
                     names=['Water Control Zone', 'Dates'], length=183)
```

```
In [31]: ss = df['Suspended Solids (mg/L)'].unstack(level = 'Water Control Zone')
print(ss)
```

Water Control Zone	Deep Bay	Eastern Buffer	Junk Bay	Mirs Bay	\
Dates					
2021-01-06	NaN	1. 9	1. 8	NaN	
2021-01-07	NaN	NaN	NaN	NaN	
2021-01-27	NaN	NaN	NaN	0. 633333	
2021-01-28	5. 7	NaN	NaN	NaN	
2021-02-03	NaN	NaN	NaN	NaN	
...	...	...	...	...	
2021-12-13	NaN	NaN	NaN	NaN	
2021-12-15	NaN	NaN	NaN	NaN	
2021-12-16	NaN	NaN	NaN	4. 438889	
2021-12-17	NaN	NaN	NaN	NaN	
2021-12-23	NaN	NaN	NaN	NaN	

Water Control Zone	North Western	Port Shelter	Southern	\
Dates				
2021-01-06	NaN	1. 000000	NaN	
2021-01-07	NaN	NaN	3. 833333	
2021-01-27	NaN	NaN	NaN	
2021-01-28	1. 5	NaN	NaN	
2021-02-03	NaN	NaN	9. 466667	
...	...	...	...	
2021-12-13	NaN	4. 362963	NaN	
2021-12-15	NaN	NaN	NaN	
2021-12-16	NaN	NaN	NaN	
2021-12-17	NaN	NaN	5. 037500	
2021-12-23	NaN	NaN	NaN	

Water Control Zone	Tolo Harbour and Channel	Victoria Harbour	Western Buffer	
Dates				
2021-01-06	NaN	2. 116667	NaN	
2021-01-07	NaN	4. 150000	2. 866667	
2021-01-27	0. 200000	NaN	NaN	
2021-01-28	NaN	NaN	NaN	
2021-02-03	NaN	9. 083333	8. 466667	
...	...	...	...	
2021-12-13	NaN	NaN	NaN	
2021-12-15	6. 173684	NaN	NaN	
2021-12-16	NaN	NaN	NaN	
2021-12-17	NaN	NaN	NaN	
2021-12-23	NaN	5. 994444	NaN	

[142 rows x 10 columns]

## Plot the Data

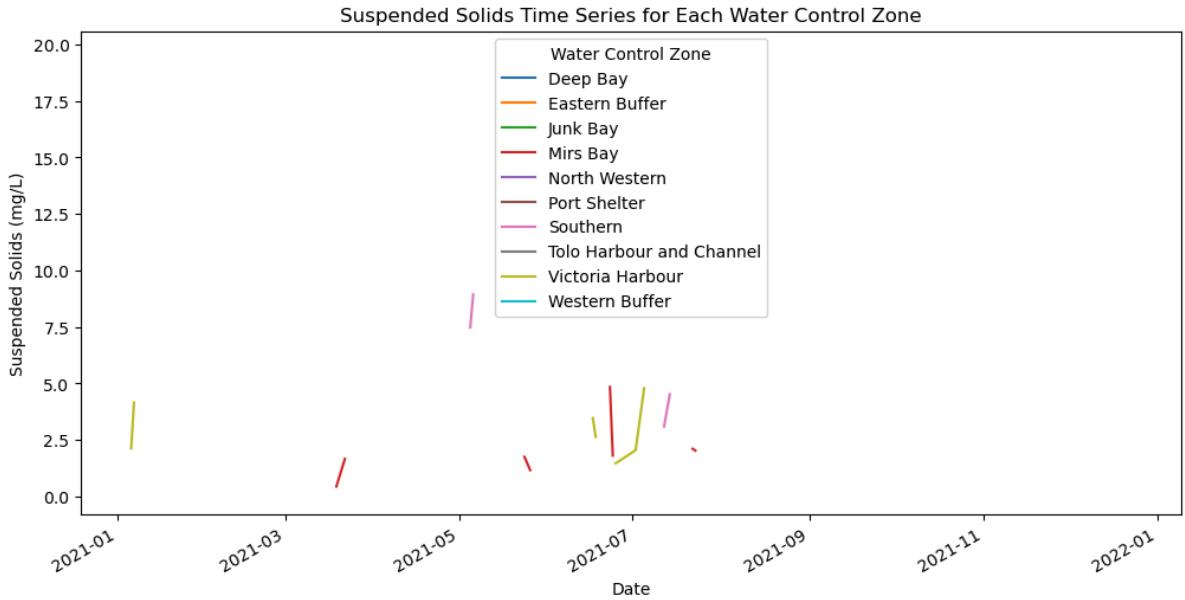
Inspecting the unstacked result, we see that there are many **missing values**. Most of these are the result of the fact that different Water Control Zones were **sampled on different dates**. The unstacked data now shows a row for every date in the suspended solids dataset, and a value is recorded for each zone and date, regardless of whether data was collected on that date.

We are interested in plotting the suspended solids time series for each zone on the same figure (a line for each zone).

Assuming your suspended solids data is now called `ss`, attempt to plot the data with `pandas` as:

```
In [32]: ss.plot(figsize=(12, 6))
plt.title("Suspended Solids Time Series for Each Water Control Zone")
plt.xlabel("Date")
plt.ylabel("Suspended Solids (mg/L)")
```

Out[32]: Text(0, 0.5, 'Suspended Solids (mg/L)')



## Adjust the Plot Legend

One obvious problem is that the **legend** is not in a very good location. We can relocate this by storing the resulting `AxesSubplot` object and modifying its associated legend information.

- `bbox_to_anchor` creates a new "bounding box" according to a tuple of spatial parameters.
  - The first two elements correspond to position (x, y) in relative axis units (0 to 1).
- `loc` specifies the legend location.

Legend info: [\(https://matplotlib.org/stable/api/legend\\_api.html\)](https://matplotlib.org/stable/api/legend_api.html)

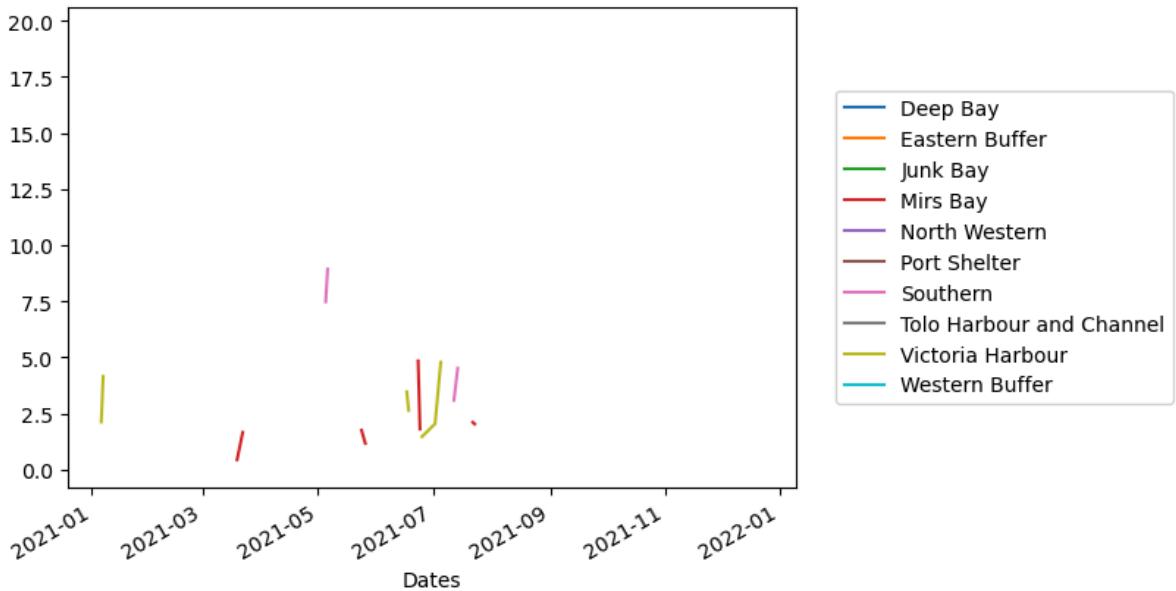
Try:

```
ss.plot().legend(bbox_to_anchor=(1.04, 0.5), loc='center left')
```

Note: If you later save this figure, it is likely that the legend will not be fully inside the plot, as it likely extends outside of the original figure plotting region. One way to account for this is with an additional parameter in `savefig()`:

```
plt.savefig('figure.png', bbox_inches="tight")
```

```
In [33]: ss.plot().legend(bbox_to_anchor=(1.04, 0.5), loc='center left')
plt.savefig('figure.png', bbox_inches='tight')
```



## Investigate the Plot Representation

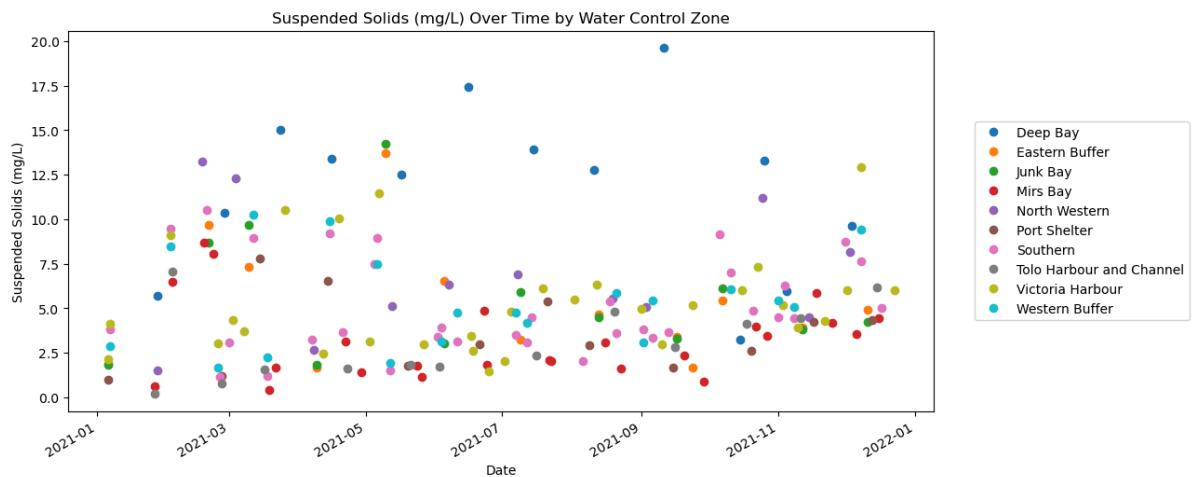
The other major problem with this plot is that very little of the data is visible. Lines are only drawn between adjacent valid numbers. Everywhere that numbers are separated by NaNs, no line is drawn.

Re-plot the data with out drawn lines, only markers.

In the resulting figure, you should see many more data points than are depicted above.

```
In [34]: ss.plot(figsize=(12, 6), lw=0, marker='o').legend(bbox_to_anchor=(1.04, 0.5), loc='center left')
plt.title('Suspended Solids (mg/L) Over Time by Water Control Zone')
plt.xlabel('Date')
plt.ylabel('Suspended Solids (mg/L)')
```

Out[34]: Text(0, 0.5, 'Suspended Solids (mg/L)')



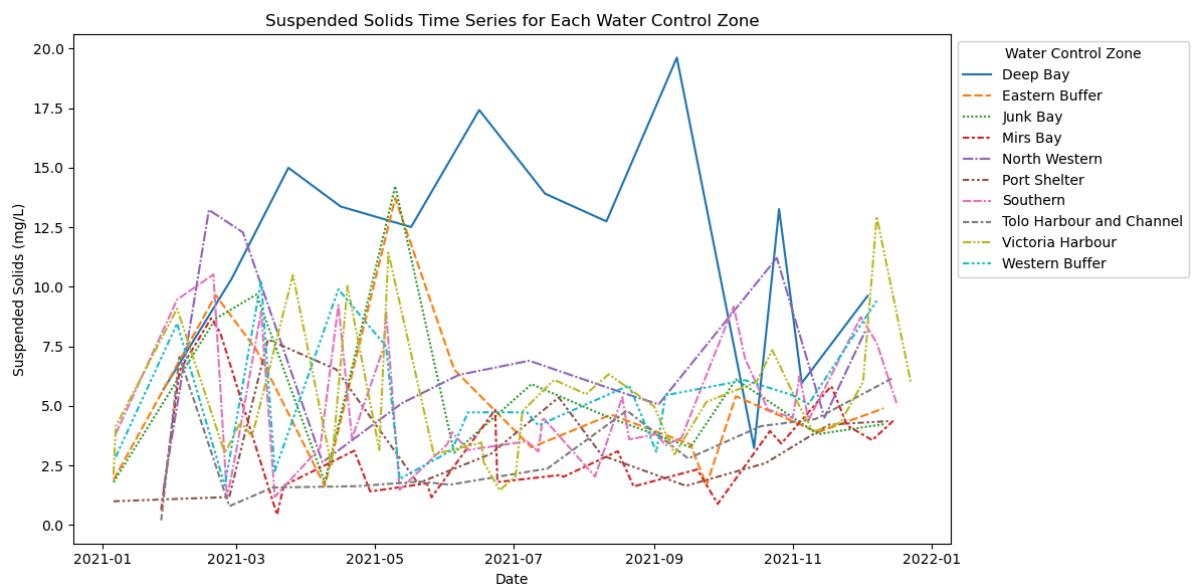
## Solution #1: seaborn

seaborn's `lineplot()` works a bit differently: it will connect data points across the NaN values. Specify the `data` parameter to test this.

- The legend can be repositioned in the same way as for pandas plotting.

```
In [35]: plt.figure(figsize=(12, 6))
sns.lineplot(data=ss)
plt.title("Suspended Solids Time Series for Each Water Control Zone")
plt.xlabel("Date")
plt.ylabel("Suspended Solids (mg/L)")

plt.legend(title="Water Control Zone", loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()
```



## Solution #2: matplotlib

As an alternative to relying on `seaborn`, we could modify the data to make similar figures with `pandas` or `matplotlib`.

One way to do this is to **mask the missing values**.

Let's try this with `matplotlib`. Create a `for` loop over the column names (zones).

In the loop:

- Isolate the values in the column.
- Isolate the index values.
- Create a mask (True / False) of finite values (non-NaNs).
- Plot a line: masked indices, versus masked values.

After the loop:

- Adjust the legend.

```
In [36]: plt.figure(figsize=(12, 6))

for zone in ss.columns:
    values = ss[zone]

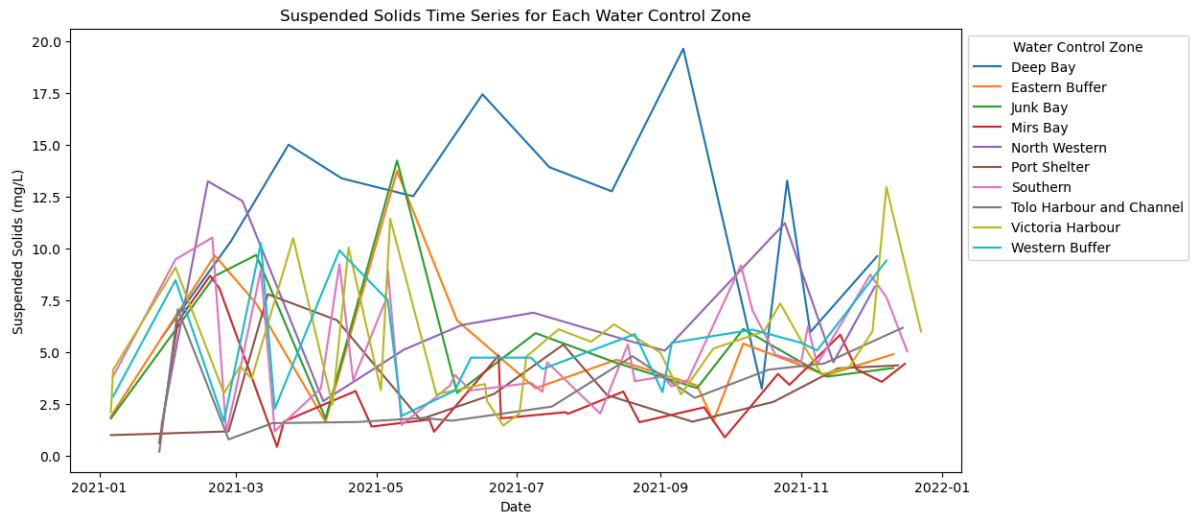
    index = ss.index
    mask = np.isfinite(values)

    plt.plot(index[mask], values[mask], label=zone)

plt.title("Suspended Solids Time Series for Each Water Control Zone")
plt.xlabel("Date")
plt.ylabel("Suspended Solids (mg/L)")

plt.legend(title="Water Control Zone", loc='upper left', bbox_to_anchor=(1, 1))
```

Out[36]: <matplotlib.legend.Legend at 0x21ee056fc20>



## Solution #3: Interpolation

Another option is to **interpolate** between the known data points to **fill in the missing values**.

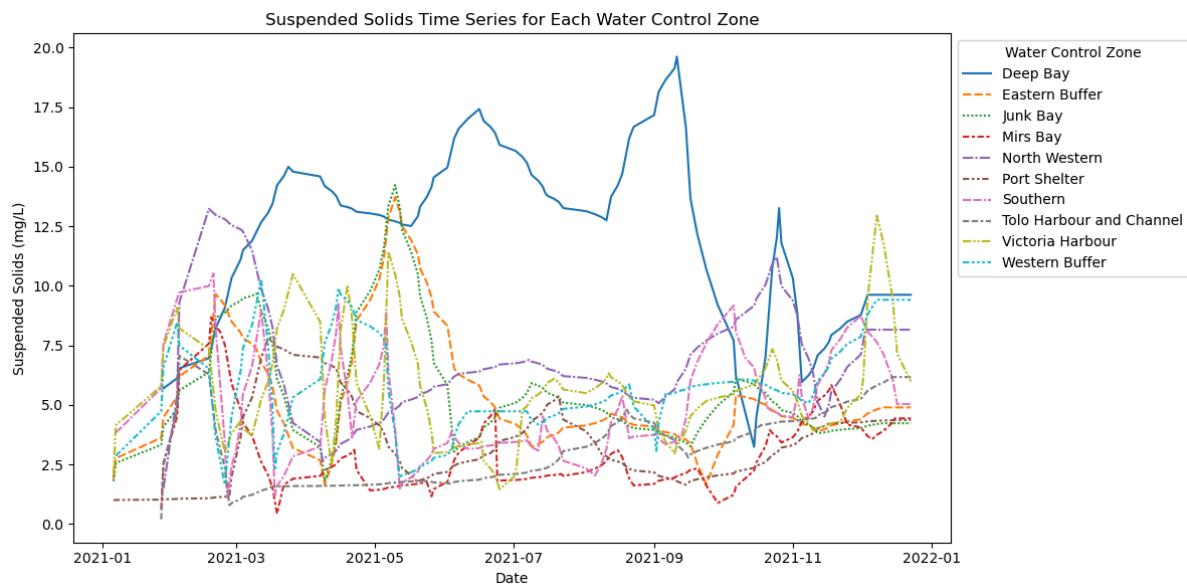
Let's try this with `seaborn`.

- To **linearly** interpolate the data in the columns, use the `pandas` method `df.interpolate()`.
  - See: [`https://pandas.pydata.org/pandas-docs/version/2.2.2/reference/api/pandas.DataFrame.interpolate.html`](https://pandas.pydata.org/pandas-docs/version/2.2.2/reference/api/pandas.DataFrame.interpolate.html) ([`https://pandas.pydata.org/pandas-docs/version/2.2.2/reference/api/pandas.DataFrame.interpolate.html`](https://pandas.pydata.org/pandas-docs/version/2.2.2/reference/api/pandas.DataFrame.interpolate.html))
  - Test this without any arguments. **Can you determine why the lines look different?**
  - If you cannot determine what has gone wrong, check the description of the `method` parameter for `interpolate()`.
    - What is happening at the latest times for some of the zones? Why do the lines become horizontal?

- Once you have sorted the issue, be sure to inspect the interpolated DataFrame to clearly understand how it has modified the data.
- You might explore additional interpolation options. For this data, few others produce reasonable results.

```
In [37]: plt.figure(figsize=(12, 6))
ss_interpolated = ss.interpolate(method='linear')
sns.lineplot(data=ss_interpolated)
plt.title("Suspended Solids Time Series for Each Water Control Zone")
plt.xlabel("Date")
plt.ylabel("Suspended Solids (mg/L)")

plt.legend(title="Water Control Zone", loc='upper left', bbox_to_anchor=(1, 1))
plt.tight_layout()
plt.show()
```



## Correlations and Heatmaps

The `./data/longley.csv` data set contains US macroeconomic variables measured over the middle of the last century. The variables are:

Column	Description
GNP.deflator	GNP implicit price deflator (a measure of inflation)
GNP	Gross National Product
Unemployed	number of unemployed
Armed.Forces	number of people in the armed forces
Population	population with age older than 14
Year	the year (1947 - 1962)
Employed	number of people employed

**Explore the data and find which variables appear to be related.**

**Notes:**

- Take a look to the **correlation** function in the following link to review the relationship between variables.
  - [\(https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html\)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.corr.html)
  - Computes a pairwise correlation, comparing all of the columns in a DataFrame against one another..
- A good way to visualise correlation comparisons is with a **heatmap**. Please take a look to the following link to learn how to present the correlation between variables with this kind of graph.
  - [\(https://seaborn.pydata.org/generated/seaborn.heatmap.html\)](https://seaborn.pydata.org/generated/seaborn.heatmap.html)
  - Plots any rectangular dataset (like that produced by pairwise correlation).
  - Customise the display:
    - cmap : [\(https://matplotlib.org/stable/tutorials/colors/colormaps.html\)](https://matplotlib.org/stable/tutorials/colors/colormaps.html) (possibly something **diverging**)
    - square : Is the data this shape?
    - annot : write the data values on the plot
    - vmin , vmax : What should the range be?
    - annot\_kws={"fontsize":????} : set the font size in the annotations (and other potential keyword arguments)
    - linecolor , linewidths : Do you want to add lines of some design?

```
In [3]: # Simple read of the data
longley = pd.read_csv('data/longley.csv')
longley
```

Out[3]:

	GNP.deflator	GNP	Unemployed	Armed.Forces	Population	Year	Employed
1947	83.0	234.289	235.6	159.0	107.608	1947	60.323
1948	88.5	259.426	232.5	145.6	108.632	1948	61.122
1949	88.2	258.054	368.2	161.6	109.773	1949	60.171
1950	89.5	284.599	335.1	165.0	110.929	1950	61.187
1951	96.2	328.975	209.9	309.9	112.075	1951	63.221
1952	98.1	346.999	193.2	359.4	113.270	1952	63.639
1953	99.0	365.385	187.0	354.7	115.094	1953	64.989
1954	100.0	363.112	357.8	335.0	116.219	1954	63.761
1955	101.2	397.469	290.4	304.8	117.388	1955	66.019
1956	104.6	419.180	282.2	285.7	118.734	1956	67.857
1957	108.4	442.769	293.6	279.8	120.445	1957	68.169
1958	110.8	444.546	468.1	263.7	121.950	1958	66.513
1959	112.6	482.704	381.3	255.2	123.366	1959	68.655
1960	114.2	502.601	393.1	251.4	125.368	1960	69.564
1961	115.7	518.173	480.6	257.2	127.852	1961	69.331
1962	116.9	554.894	400.7	282.7	130.081	1962	70.551

```
In [4]: # Compute the correlation matrix
corr_matrix = longley.corr()

# Plot the heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix,
            annot=True, # Display the correlation coefficients in each cell
            cmap='coolwarm', # Use a diverging colormap (coolwarm) for contrast between
            square=True, # Ensure the heatmap is square-shaped
            vmin=0, vmax=1, # Set the color scale limits from -1 to 1
            annot_kws={"fontsize": 10}, # Set the font size of the annotations
            linewidths=0.5, # Set the width of the lines separating the cells
            linecolor='black') # Set the color of the lines separating the cells

# Add title
plt.title("Correlation Heatmap of Longley Dataset")

# Display the heatmap
plt.tight_layout() # Adjust the layout to ensure everything fits
plt.show()
```

