In [10]:

```python
# Import pandas
import pandas as pd

# Read the file into a DataFrame: df
airquality = pd.read_csv('airquality.csv')

airquality.head()
```

Out[10]:

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.0	7.4	67	5	1
1	36.0	118.0	8.0	72	5	2
2	12.0	149.0	12.6	74	5	3
3	18.0	313.0	11.5	62	5	4
4	NaN	NaN	14.3	56	5	5

Dropping Row/Columns

In [24]:

```python
#dropping first row
airquality.drop(0)
```

Out[24]:

	Ozone	Solar.R	Wind	Temp	Month	Day
1	36.0	118.0	8.0	72	5	2
2	12.0	149.0	12.6	74	5	3
3	18.0	313.0	11.5	62	5	4
4	NaN	NaN	14.3	56	5	5
5	28.0	NaN	14.9	66	5	6
...	...	...	...	...	...	...
148	30.0	193.0	6.9	70	9	26
149	NaN	145.0	13.2	77	9	27
150	14.0	191.0	14.3	75	9	28
151	18.0	131.0	8.0	76	9	29
152	20.0	223.0	11.5	68	9	30

152 rows × 6 columns

In [27]:

```python
#dropping set of specific rows
airquality.drop(index=[1,2,4,6])
```

Out[27]:

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.0	7.4	67	5	1
3	18.0	313.0	11.5	62	5	4
5	28.0	NaN	14.9	66	5	6
7	19.0	99.0	13.8	59	5	8
8	8.0	19.0	20.1	61	5	9
...	...	...	...	...	...	...
148	30.0	193.0	6.9	70	9	26
149	NaN	145.0	13.2	77	9	27
150	14.0	191.0	14.3	75	9	28
151	18.0	131.0	8.0	76	9	29
152	20.0	223.0	11.5	68	9	30

149 rows × 6 columns

In [29]:

```
#dropping columns
airquality.drop(['Temp'],axis=1)
```

Out[29]:

	Ozone	Solar.R	Wind	Month	Day
0	41.0	190.0	7.4	5	1
1	36.0	118.0	8.0	5	2
2	12.0	149.0	12.6	5	3
3	18.0	313.0	11.5	5	4
4	NaN	NaN	14.3	5	5
...	...	...	...	...	...
148	30.0	193.0	6.9	9	26
149	NaN	145.0	13.2	9	27
150	14.0	191.0	14.3	9	28
151	18.0	131.0	8.0	9	29
152	20.0	223.0	11.5	9	30

153 rows × 5 columns

In [35]:

```
airquality.drop(['Wind','Month'], axis=1)
```

Out[35]:

	Ozone	Solar.R	Temp	Day
0	41.0	190.0	67	1
1	36.0	118.0	72	2
2	12.0	149.0	74	3
3	18.0	313.0	62	4
4	NaN	NaN	56	5
...	...	...	...	...
148	30.0	193.0	70	26
149	NaN	145.0	77	27
150	14.0	191.0	75	28
151	18.0	131.0	76	29
152	20.0	223.0	68	30

153 rows × 4 columns

Working with missing Values

In [37]:

```
airquality.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 153 entries, 0 to 152
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Ozone    116 non-null    float64
 1   Solar.R  146 non-null    float64
 2   Wind     153 non-null    float64
 3   Temp     153 non-null    int64  
 4   Month    153 non-null    int64  
 5   Day      153 non-null    int64  
dtypes: float64(3), int64(3)
memory usage: 7.3 KB
```

In [38]:

```
#fill with zero
airquality.fillna(0)
```

Out[38]:

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.0	7.4	67	5	1
1	36.0	118.0	8.0	72	5	2
2	12.0	149.0	12.6	74	5	3
3	18.0	313.0	11.5	62	5	4
4	0.0	0.0	14.3	56	5	5
...	...	...	...	...	...	...
148	30.0	193.0	6.9	70	9	26
149	0.0	145.0	13.2	77	9	27
150	14.0	191.0	14.3	75	9	28
151	18.0	131.0	8.0	76	9	29
152	20.0	223.0	11.5	68	9	30

153 rows × 6 columns

In [40]:
```python
#fill with mean
airquality.Ozone.fillna(airquality.Ozone.mean())
```

Out[40]:
```
0      41.00000
1      36.00000
2      12.00000
3      18.00000
4      42.12931
         ...
148    30.00000
149    42.12931
150    14.00000
151    18.00000
152    20.00000
Name: Ozone, Length: 153, dtype: float64
```

In [42]:
```python
# bfill/ffill
airquality.fillna(method='bfill')
airquality.fillna(method='ffill')
```

Out[42]:

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.0	7.4	67	5	1
1	36.0	118.0	8.0	72	5	2
2	12.0	149.0	12.6	74	5	3
3	18.0	313.0	11.5	62	5	4
4	18.0	313.0	14.3	56	5	5
...	...	...	...	...	...	...
148	30.0	193.0	6.9	70	9	26
149	30.0	145.0	13.2	77	9	27
150	14.0	191.0	14.3	75	9	28
151	18.0	131.0	8.0	76	9	29
152	20.0	223.0	11.5	68	9	30

153 rows × 6 columns

In [57]:
```python
airquality[:6]
```

Out[57]:

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.0	7.4	67	5	1
1	36.0	118.0	8.0	72	5	2
2	12.0	149.0	12.6	74	5	3
3	18.0	313.0	11.5	62	5	4
4	NaN	NaN	14.3	56	5	5
5	28.0	NaN	14.9	66	5	6

In [55]:
```python
#fill using interolation
airquality.interpolate(method ='linear', limit_direction ='forward')
```

Out[55]:

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.000000	7.4	67	5	1
1	36.0	118.000000	8.0	72	5	2
2	12.0	149.000000	12.6	74	5	3
3	18.0	313.000000	11.5	62	5	4
4	23.0	308.333333	14.3	56	5	5
...	...	...	...	...	...	...
148	30.0	193.000000	6.9	70	9	26
149	22.0	145.000000	13.2	77	9	27
150	14.0	191.000000	14.3	75	9	28
151	18.0	131.000000	8.0	76	9	29
152	20.0	223.000000	11.5	68	9	30

methods : {'linear', 'time', 'index', 'values', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'barycentric', 'krogh', 'polynomial', 'spline', 'piecewise_polynomial', 'from_derivatives', 'pchip', 'akima'}

Linear interpolation is a method of estimating values between two known values in a series of data. In the context of filling missing values in a pandas DataFrame, linear interpolation estimates the missing values by computing a straight line between the two nearest known values. The method assumes that the change in the dependent variable (the missing value) is constant with respect to the independent variable (time or index). The missing value is then estimated as a weighted average of the two nearest known values, where the weight is proportional to the distance between the missing value and the known values.

In [58]:
```
airquality.interpolate(method ='quadratic', limit_direction ='backward')
```
Out[58]:

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.000000	190.00000	7.4	67	5	1
1	36.000000	118.00000	8.0	72	5	2
2	12.000000	149.00000	12.6	74	5	3
3	18.000000	313.00000	11.5	62	5	4
4	26.137476	413.21389	14.3	56	5	5
...	...	...	...	...	...	...
148	30.000000	193.00000	6.9	70	9	26
149	24.224969	145.00000	13.2	77	9	27
150	14.000000	191.00000	14.3	75	9	28
151	18.000000	131.00000	8.0	76	9	29
152	20.000000	223.00000	11.5	68	9	30

In [64]:
```
airquality.interpolate(method ='nearest', limit_direction ='forward')
```
Out[64]:

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.0	7.4	67	5	1
1	36.0	118.0	8.0	72	5	2
2	12.0	149.0	12.6	74	5	3
3	18.0	313.0	11.5	62	5	4
4	18.0	313.0	14.3	56	5	5
...	...	...	...	...	...	...
148	30.0	193.0	6.9	70	9	26
149	30.0	145.0	13.2	77	9	27
150	14.0	191.0	14.3	75	9	28
151	18.0	131.0	8.0	76	9	29
152	20.0	223.0	11.5	68	9	30

153 rows × 6 columns

In [71]:
```
airquality[:6]
```
Out[71]:

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.0	190.0	7.4	67	5	1
1	36.0	118.0	8.0	72	5	2
2	12.0	149.0	12.6	74	5	3
3	18.0	313.0	11.5	62	5	4
4	NaN	NaN	14.3	56	5	5
5	28.0	NaN	14.9	66	5	6

In [74]:
```
airquality.interpolate(method ='quadratic', limit_direction ='backward', limit = 1)[:6]
```
Out[74]:

	Ozone	Solar.R	Wind	Temp	Month	Day
0	41.000000	190.000000	7.4	67	5	1
1	36.000000	118.000000	8.0	72	5	2
2	12.000000	149.000000	12.6	74	5	3
3	18.000000	313.000000	11.5	62	5	4
4	26.137476	NaN	14.3	56	5	5
5	28.000000	412.882252	14.9	66	5	6

The choice of interpolation method depends on several factors, including the nature of the data, the desired accuracy, and the computational resources available. Here are some general guidelines on when to use each method:

Linear Interpolation: Simple and fast, best used for small datasets or when the relationship between the variables is roughly linear. Can also be used when computational resources are limited.

Polynomial Interpolation: Can capture non-linear relationships, but is more complex than linear interpolation. Good for datasets where the relationship between the variables is well understood and can be described by a polynomial function.

Spline Interpolation: Flexible and can capture non-linear relationships. The cubic spline is the most commonly used form of spline interpolation and is good for datasets where the relationship between the variables is complex.

Kriging: A type of spatial interpolation used in geostatistics, it models the spatial autocorrelation of the data to make predictions. Good for geospatial datasets where the relationship between the variables is influenced by the spatial location.

Radial Basis Function Interpolation: A non-parametric method, it is well suited for high-dimensional data and can capture complex relationships. Good for datasets where the relationship between the variables is difficult to describe or is unknown.

It's also possible to use a combination of different interpolation methods for a single problem, depending on the specific requirements and constraints.

Droping NANs

In [75]:
```python
df = pd.DataFrame({"A":[12, 4, 5, None, 1],
                   "B":[None, 2, 54, 3, None],
                   "C":[20, 16, None, 3, 8],
                   "D":[14, 3, None, None, 6]})
```

In [76]:
```python
df
```

Out[76]:

	A	B	C	D
0	12.0	NaN	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	NaN	NaN
3	NaN	3.0	3.0	NaN
4	1.0	NaN	8.0	6.0

In [77]:
```python
df.dropna()
```

Out[77]:

	A	B	C	D
1	4.0	2.0	16.0	3.0

In [78]:
```python
df.dropna(axis=1)
```

Out[78]:

0
1
2
3
4

In [82]:
```python
# Drop any row that has at least 3 NON-NaNs within it:
df.dropna(axis=0, thresh=3)
```

Out[82]:

	A	B	C	D
0	12.0	NaN	20.0	14.0
1	4.0	2.0	16.0	3.0
4	1.0	NaN	8.0	6.0

In [84]:
```python
#droping duplicates
df.drop_duplicates()
```

Out[84]:

	A	B	C	D
0	12.0	NaN	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	NaN	NaN
3	NaN	3.0	3.0	NaN
4	1.0	NaN	8.0	6.0

In [88]:
```python
import numpy as np
df.append({'A':1.0,'B':np.nan,'C':8.0,'D':6.0},ignore_index=True).drop_duplicates()
```

Out[88]:

	A	B	C	D
0	12.0	NaN	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	NaN	NaN
3	NaN	3.0	3.0	NaN
4	1.0	NaN	8.0	6.0

In [93]:
```python
df.append({'A':2.0,'B':5.0,'C':8.0,'D':6.0},ignore_index=True).drop_duplicates(subset=['C', 'D'])
```

Out[93]:

	A	B	C	D
0	12.0	NaN	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	NaN	NaN
3	NaN	3.0	3.0	NaN
4	1.0	NaN	8.0	6.0

In [95]:

```
#The drop=True parameter tells Pandas not to keep a backup copy of the original index.
df.reset_index(drop=True)
```

Out[95]:

	A	B	C	D
0	12.0	NaN	20.0	14.0
1	4.0	2.0	16.0	3.0
2	5.0	54.0	NaN	NaN
3	NaN	3.0	3.0	NaN
4	1.0	NaN	8.0	6.0