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

# Load the dataset

df = pd.read_csv("/content/delhi_aqi.csv")
```

```python
# 1. Working with Series and DataFrame
print("First few rows of the DataFrame:")
print(df.head())
```

```
First few rows of the DataFrame:
                  date       co     no     no2     o3    so2   pm2_5    pm10  \
0  2020-11-25 01:00:00  2616.88   2.18   70.60  13.59  38.62  364.61  411.73   
1  2020-11-25 02:00:00  3631.59  23.25   89.11   0.33  54.36  420.96  486.21   
2  2020-11-25 03:00:00  4539.49  52.75  100.08   1.11  68.67  463.68  541.95   
3  2020-11-25 04:00:00  4539.49  50.96  111.04   6.44  78.20  454.81  534.00   
4  2020-11-25 05:00:00  4379.27  42.92  117.90  17.17  87.74  448.14  529.19   

     nh3  
0  28.63  
1  41.04  
2  49.14  
3  48.13  
4  46.61  
```

```python
# Convert one column to a Pandas Series
print(df.columns)# printing all the columns
aqi_series = df['date']

print("Series object from date column:")
print(aqi_series.head())
```

```
Index(['date', 'co', 'no', 'no2', 'o3', 'so2', 'pm2_5', 'pm10', 'nh3'], dtype='object')
Series object from date column:
0    2020-11-25 01:00:00
1    2020-11-25 02:00:00
2    2020-11-25 03:00:00
3    2020-11-25 04:00:00
4    2020-11-25 05:00:00
Name: date, dtype: object
```

```python
# 2. Indexing and selecting data
print("Selecting a specific column:")
print(df['pm2_5'].head())  # Selecting a single column

print("Selecting multiple columns:")
print(df[['pm2_5', 'pm10']].head())  # Selecting multiple columns

print("Selecting rows using loc and iloc:")
print(df.loc[0])  # Selecting first row by label
print(df.iloc[0])  # Selecting first row by index
```

```
Selecting a specific column:
0    364.61
1    420.96
2    463.68
3    454.81
4    448.14
Name: pm2_5, dtype: float64
Selecting multiple columns:
    pm2_5    pm10
0  364.61  411.73
1  420.96  486.21
2  463.68  541.95
3  454.81  534.00
4  448.14  529.19
Selecting rows using loc and iloc:
date     2020-11-25 01:00:00
co                   2616.88
no                      2.18
no2                     70.6
o3                     13.59
so2                    38.62
pm2_5                 364.61
pm10                  411.73
nh3                    28.63
Name: 0, dtype: object
date     2020-11-25 01:00:00
co                   2616.88
no                      2.18
no2                     70.6
o3                     13.59
```

```
so2                   38.62
pm2_5                364.61
pm10                 411.73
nh3                   28.63
Name: 0, dtype: object
```

```
# 3. Using universal functions for index preservation
data = df.select_dtypes(include=[np.number]).values
square_root = np.sqrt(data)  # Square root of each element
log_values = np.log1p(data)  # Natural log (log(1 + x) to avoid log(0) errors)
exponential = np.exp(data)  # Exponential function
absolute_values = np.abs(data)  # Absolute values

print("Square Root:")
print(square_root)
print("Natural Log:")
print(log_values)
print("Exponential Function:")
print(exponential)
print("Absolute Values:")
print(absolute_values)
```

```
Square Root:
[[51.15544937  1.47648231  8.40238062 ...  6.21449918 20.29113107
   5.35070089]
 [60.26267502  4.82182538  9.43980932 ...  7.37292344 22.05017007
   6.40624695]
 [67.37573747  7.26291952 10.0039992  ...  8.28673639 23.27981959
   7.00999287]
 ...
 [43.8475769   2.85657137  6.33245608 ...  6.58710862 17.20668475
   3.54118624]
 [36.90325189  3.00832179  7.26498451 ... 10.00699755 13.84990975
   2.73313007]
 [33.68783163  2.93428015  7.54254599 ... 10.51807967 11.84314148
   2.34733892]]
Natural Log:
[[7.87012011 1.1568812  4.27109507 ... 3.67933404 6.02279363 3.38878736]
 [8.19770117 3.18841662 4.50103115 ... 4.01385731 6.18869524 3.73862155]
 [8.42079021 3.98434367 4.61591228 ... 4.24376981 6.29701723 3.91481909]
 ...
 [7.56195891 2.21484618 3.71600812 ... 3.79301422 5.6939678  2.60564827]
 [7.21733337 2.30757263 3.98490165 ... 4.6165057  5.26175711 2.13653051]
 [7.03515416 2.26280422 4.05854466 ... 4.71518983 4.95060216 1.87333946]]
Exponential Function:
[[           inf 8.84630626e+000 4.58342809e+030 ... 5.92178848e+016
  6.48734559e+178 2.71550756e+012]
 [           inf 1.25125753e+010 5.01165633e+038 ... 4.05740197e+023
  1.43985927e+211 6.65956002e+017]
 [           inf 8.11024400e+022 2.91200254e+043 ... 6.65276331e+029
  2.32217245e+235 2.19397079e+021]
 ...
 [           inf 3.49818660e+003 2.60140951e+017 ... 6.98292809e+018
  3.81563881e+128 2.79288339e+005]
 [           inf 8.51853792e+003 8.35723770e+022 ... 3.09207072e+043
  2.02473187e+083 1.75460669e+003]
 [           inf 5.48624868e+003 5.09346206e+024 ... 1.11172797e+048
  8.20623655e+060 2.47151127e+002]]
Absolute Values:
[[2.61688e+03 2.18000e+00 7.06000e+01 ... 3.86200e+01 4.11730e+02
  2.86300e+01]
 [3.63159e+03 2.32500e+01 8.91100e+01 ... 5.43600e+01 4.86210e+02
  4.10400e+01]
 [4.53949e+03 5.27500e+01 1.00080e+02 ... 6.86700e+01 5.41950e+02
  4.91400e+01]
 ...
 [1.92261e+03 8.16000e+00 4.01000e+01 ... 4.33900e+01 2.96070e+02
  1.25400e+01]
 [1.36185e+03 9.05000e+00 5.27800e+01 ... 1.00140e+02 1.91820e+02
  7.47000e+00]
 [1.13487e+03 8.61000e+00 5.68900e+01 ... 1.10630e+02 1.40260e+02
  5.51000e+00]]
<ipython-input-26-f6c6ee5a769a>:5: RuntimeWarning: overflow encountered in exp
  exponential = np.exp(data)  # Exponential function
```

```
# 4. Index alignment and operations between Series and DataFrames
mean_values = df.mean(numeric_only=True)
print("Subtracting mean from each column:")
print(df.subtract(mean_values))
```

```
Subtracting mean from each column:
                co date        nh3         no        no2         o3  \
0      -312.348628  NaN   3.520185 -31.480702   4.378701 -46.756239
1       702.361372  NaN  15.930185 -10.410702  22.888701 -60.016239
2      1610.261372  NaN  24.030185  19.089298  33.858701 -59.236239
3      1610.261372  NaN  23.020185  17.299298  44.818701 -53.906239
4      1450.041372  NaN  21.500185   9.259298  51.678701 -43.176239
```

```
...              ...  ...        ...        ...        ...        ...
18771 -1166.838628  NaN -18.839815 -29.020702 -29.211299 -27.086239
18772 -1193.538628  NaN -15.989815 -26.840702 -31.261299 -13.856239
18773 -1006.618628  NaN -12.569815 -25.500702 -26.121299  -3.836239
18774 -1567.378628  NaN -17.639815 -24.610702 -13.441299  11.183761
18775 -1794.358628  NaN -19.599815 -25.050702  -9.331299  19.763761

             pm10       pm2_5        so2
0      111.637034  126.479691 -28.073633
1      186.117034  182.829691 -12.333633
2      241.857034  225.549691   1.976367
3      233.907034  216.679691  11.506367
4      229.097034  210.009691  21.046367
...           ...         ...        ...
18771  -10.252966   -6.980309 -36.173633
18772  -19.572966  -13.050309 -32.363633
18773   -4.022966    4.359691 -23.303633
18774 -108.272966  -72.460309  33.446367
18775 -159.832966 -114.370309  43.936367

[18776 rows x 9 columns]
```

```python
# 5. Handling missing data
df.fillna(df.mean(numeric_only=True), inplace=True)  # Fill missing values with mean
print("Missing values handled using mean replacement.")
```

```
Missing values handled using mean replacement.
```

```python
# 6. Operating on null values
print("Checking for null values:")
print(df.isnull().sum())  # Count of missing values per column
```

```
Checking for null values:
date     0
co       0
no       0
no2      0
o3       0
so2      0
pm2_5    0
pm10     0
nh3      0
dtype: int64
```

```python
# 7. Hierarchical Indexing
df.set_index(['date', 'pm2_5'], inplace=True)
print("DataFrame after setting hierarchical index:")
print(df.head())
```

```
DataFrame after setting hierarchical index:
                                co     no     no2     o3    so2    pm10  \
date                pm2_5
2020-11-25 01:00:00 364.61  2616.88   2.18   70.60  13.59  38.62  411.73
2020-11-25 02:00:00 420.96  3631.59  23.25   89.11   0.33  54.36  486.21
2020-11-25 03:00:00 463.68  4539.49  52.75  100.08   1.11  68.67  541.95
2020-11-25 04:00:00 454.81  4539.49  50.96  111.04   6.44  78.20  534.00
2020-11-25 05:00:00 448.14  4379.27  42.92  117.90  17.17  87.74  529.19

                              nh3
date                pm2_5
2020-11-25 01:00:00 364.61  28.63
2020-11-25 02:00:00 420.96  41.04
2020-11-25 03:00:00 463.68  49.14
2020-11-25 04:00:00 454.81  48.13
2020-11-25 05:00:00 448.14  46.61
```