```
import numpy as np
import pandas as pd

# Load the dataset

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

# Convert numerical columns to a NumPy array (ignoring the 'date' column)
data = df.iloc[:, 1:].values

```

```
# 1. Checking the data type using dtype parameter
print("Data Type of the array:", data.dtype)
```

```
Data Type of the array: float64
```

```
# 2. creating a random array using np.random
random_array = np.random.rand(5, 5)
print("Random Array:", random_array)
```

```
Random Array: [[0.00366381 0.68352303 0.98639453 0.20297084 0.86127073]
 [0.44095534 0.55866181 0.21768111 0.00299818 0.91356912]
 [0.73685697 0.63090312 0.23735596 0.04475376 0.14801059]
 [0.38601212 0.55333576 0.59490042 0.27746828 0.00805554]
 [0.5975309  0.86026083 0.40762127 0.71730599 0.05998878]]
```

```
# 3. Indexing
print("First row of data:", data[0])  # Grab the first row
print("Element at row 1, column 2:", data[0, 1])  # Get a specific element

```

```
First row of data: [2.61688e+03 2.18000e+00 7.06000e+01 1.35900e+01 3.86200e+01 3.64610e+02
 4.11730e+02 2.86300e+01]
Element at row 1, column 2: 2.18
```

```
# 4. Slicing
print("First 5 rows:", data[:5])  # Get the first 5 rows
print("First 3 columns:", data[:, :3])  # Get the first 3 columns
```

```
First 5 rows: [[2.61688e+03 2.18000e+00 7.06000e+01 1.35900e+01 3.86200e+01 3.64610e+02
  4.11730e+02 2.86300e+01]
 [3.63159e+03 2.32500e+01 8.91100e+01 3.30000e-01 5.43600e+01 4.20960e+02
  4.86210e+02 4.10400e+01]
 [4.53949e+03 5.27500e+01 1.00080e+02 1.11000e+00 6.86700e+01 4.63680e+02
  5.41950e+02 4.91400e+01]
 [4.53949e+03 5.09600e+01 1.11040e+02 6.44000e+00 7.82000e+01 4.54810e+02
  5.34000e+02 4.81300e+01]
 [4.37927e+03 4.29200e+01 1.17900e+02 1.71700e+01 8.77400e+01 4.48140e+02
  5.29190e+02 4.66100e+01]]
First 3 columns: [[2.61688e+03 2.18000e+00 7.06000e+01]
 [3.63159e+03 2.32500e+01 8.91100e+01]
 [4.53949e+03 5.27500e+01 1.00080e+02]
 ...
 [1.92261e+03 8.16000e+00 4.01000e+01]
 [1.36185e+03 9.05000e+00 5.27800e+01]
 [1.13487e+03 8.61000e+00 5.68900e+01]]
```

```
# 5. Reshaping the first 25 elements into a 3D array
reshaped_array = data[:25].reshape(5, 5, -1)
print("Reshaped array shape:", reshaped_array.shape)
```

```
Reshaped array shape: (5, 5, 8)
```

```
# 6. Splitting the array into two halves and then putting them back together
split1, split2 = np.split(data, 2)
concatenated_array = np.concatenate((split1, split2))
print(split1)
print(split2)
print(concatenated_array)
```

```
[[2.61688e+03 2.18000e+00 7.06000e+01 ... 3.64610e+02 4.11730e+02
  2.86300e+01]
 [3.63159e+03 2.32500e+01 8.91100e+01 ... 4.20960e+02 4.86210e+02
  4.10400e+01]
 [4.53949e+03 5.27500e+01 1.00080e+02 ... 4.63680e+02 5.41950e+02
  4.91400e+01]
 ...
 [6.51550e+03 1.32320e+02 5.96300e+01 ... 6.13260e+02 7.11770e+02
  1.79900e+01]
 [8.01086e+03 1.96700e+02 6.92300e+01 ... 6.79080e+02 7.89700e+02
```

```
        1.77300e+01]
       [6.67572e+03 1.43050e+02 8.63700e+01 ... 5.75550e+02 6.63860e+02
        2.02700e+01]]
      [[7.37000e+03 1.37690e+02 1.20640e+02 ... 6.37630e+02 7.40850e+02
        3.34400e+01]
       [7.79724e+03 9.29800e+01 1.80960e+02 ... 7.06740e+02 8.19320e+02
        4.25600e+01]
       [2.72369e+03 1.49800e+01 9.73300e+01 ... 3.28270e+02 3.65850e+02
        1.60900e+01]
       ...
       [1.92261e+03 8.16000e+00 4.01000e+01 ... 2.42490e+02 2.96070e+02
        1.25400e+01]
       [1.36185e+03 9.05000e+00 5.27800e+01 ... 1.65670e+02 1.91820e+02
        7.47000e+00]
       [1.13487e+03 8.61000e+00 5.68900e+01 ... 1.23760e+02 1.40260e+02
        5.51000e+00]]
      [[2.61688e+03 2.18000e+00 7.06000e+01 ... 3.64610e+02 4.11730e+02
        2.86300e+01]
       [3.63159e+03 2.32500e+01 8.91100e+01 ... 4.20960e+02 4.86210e+02
        4.10400e+01]
       [4.53949e+03 5.27500e+01 1.00080e+02 ... 4.63680e+02 5.41950e+02
        4.91400e+01]
       ...
       [1.92261e+03 8.16000e+00 4.01000e+01 ... 2.42490e+02 2.96070e+02
        1.25400e+01]
       [1.36185e+03 9.05000e+00 5.27800e+01 ... 1.65670e+02 1.91820e+02
        7.47000e+00]
       [1.13487e+03 8.61000e+00 5.68900e+01 ... 1.23760e+02 1.40260e+02
        5.51000e+00]]
```

```
# 7. Some basic math on the dataset
print("Column-wise Mean:", np.mean(data, axis=0))
print("Column-wise Sum:", np.sum(data, axis=0))
```

```
Column-wise Mean: [2929.2286275    33.66070196   66.221299     60.34623935   66.69363283
  238.1303089   300.09296602   25.10981519]
Column-wise Sum: [54999196.71000001   632013.34        1243371.11        1133060.99
  1252239.65        4471134.68        5634545.53         471461.89      ]
```

```
# 8. Broadcasting array
offset = np.array([1, 2, 3, 4, 5, 6, 7, 8])
broadcasted_array = data[:5] + offset
print("Array after broadcasting:", broadcasted_array)
```

```
Array after broadcasting: [[2.61788e+03 4.18000e+00 7.36000e+01 1.75900e+01 4.36200e+01 3.70610e+02
  4.18730e+02 3.66300e+01]
 [3.63259e+03 2.52500e+01 9.21100e+01 4.33000e+00 5.93600e+01 4.26960e+02
  4.93210e+02 4.90400e+01]
 [4.54049e+03 5.47500e+01 1.03080e+02 5.11000e+00 7.36700e+01 4.69680e+02
  5.48950e+02 5.71400e+01]
 [4.54049e+03 5.29600e+01 1.14040e+02 1.04400e+01 8.32000e+01 4.60810e+02
  5.41000e+02 5.61300e+01]
 [4.38027e+03 4.49200e+01 1.20900e+02 2.11700e+01 9.27400e+01 4.54140e+02
  5.36190e+02 5.46100e+01]]
```

```
# 9. Finding rows where PM2.5 is higher than 100
high_pm2_5 = data[:, 5] > 100
print("Rows where PM2.5 is above 100:", data[high_pm2_5])
```

```
Rows where PM2.5 is above 100: [[2.61688e+03 2.18000e+00 7.06000e+01 ... 3.64610e+02 4.11730e+02
  2.86300e+01]
 [3.63159e+03 2.32500e+01 8.91100e+01 ... 4.20960e+02 4.86210e+02
  4.10400e+01]
 [4.53949e+03 5.27500e+01 1.00080e+02 ... 4.63680e+02 5.41950e+02
  4.91400e+01]
 ...
 [1.92261e+03 8.16000e+00 4.01000e+01 ... 2.42490e+02 2.96070e+02
  1.25400e+01]
 [1.36185e+03 9.05000e+00 5.27800e+01 ... 1.65670e+02 1.91820e+02
  7.47000e+00]
 [1.13487e+03 8.61000e+00 5.68900e+01 ... 1.23760e+02 1.40260e+02
  5.51000e+00]]
```

```
# 10. Picking specific rows using fancy indexing
indices = [0, 2, 4]
print("Selected rows:", data[indices])
```

```
Selected rows: [[2.61688e+03 2.18000e+00 7.06000e+01 1.35900e+01 3.86200e+01 3.64610e+02
  4.11730e+02 2.86300e+01]
 [4.53949e+03 5.27500e+01 1.00080e+02 1.11000e+00 6.86700e+01 4.63680e+02
  5.41950e+02 4.91400e+01]
 [4.37927e+03 4.29200e+01 1.17900e+02 1.71700e+01 8.77400e+01 4.48140e+02
  5.29190e+02 4.66100e+01]]
```

```
# 11. Sorting data in different ways
```

```
sorted_data = np.sort(data, axis=0)  # Sort each column individually
argsorted_indices = np.argsort(data[:, 0])  # Indices that sort the first column
partial_sorted = np.partition(data[:, 0], 5)[:5]  # Get the 5 smallest elements from the first column
print(argsorted_indices)
print(partial_sorted)
print(sorted_data)
```

```
[13449 13448 13303 ...  8561 17176  8560]
[260.35 263.69 267.03 270.37 270.37]
[[2.603500e+02 0.000000e+00 4.280000e+00 ... 1.183000e+01 1.507000e+01
  0.000000e+00]
 [2.636900e+02 0.000000e+00 4.330000e+00 ... 1.259000e+01 1.521000e+01
  0.000000e+00]
 [2.670300e+02 0.000000e+00 4.540000e+00 ... 1.261000e+01 1.574000e+01
  0.000000e+00]
 ...
 [2.072144e+04 4.792200e+02 4.441700e+02 ... 1.599590e+03 1.836560e+03
  2.715600e+02]
 [2.072144e+04 5.006800e+02 4.441700e+02 ... 1.627520e+03 1.870690e+03
  2.756100e+02]
 [2.114868e+04 5.006800e+02 4.606200e+02 ... 1.708090e+03 1.969930e+03
  2.877700e+02]]
```

```
# 12. Creating a structured array with column names
dtype = [('co', 'f8'), ('no', 'f8'), ('no2', 'f8')]
structured_array = np.array(list(zip(data[:, 0], data[:, 1], data[:, 2])), dtype=dtype)
print("Structured Array:", structured_array)
```

```
Structured Array: [(2616.88,  2.18,  70.6 ) (3631.59, 23.25,  89.11)
 (4539.49, 52.75, 100.08) ... (1922.61,  8.16,  40.1 )
 (1361.85,  9.05,  52.78) (1134.87,  8.61,  56.89)]
```