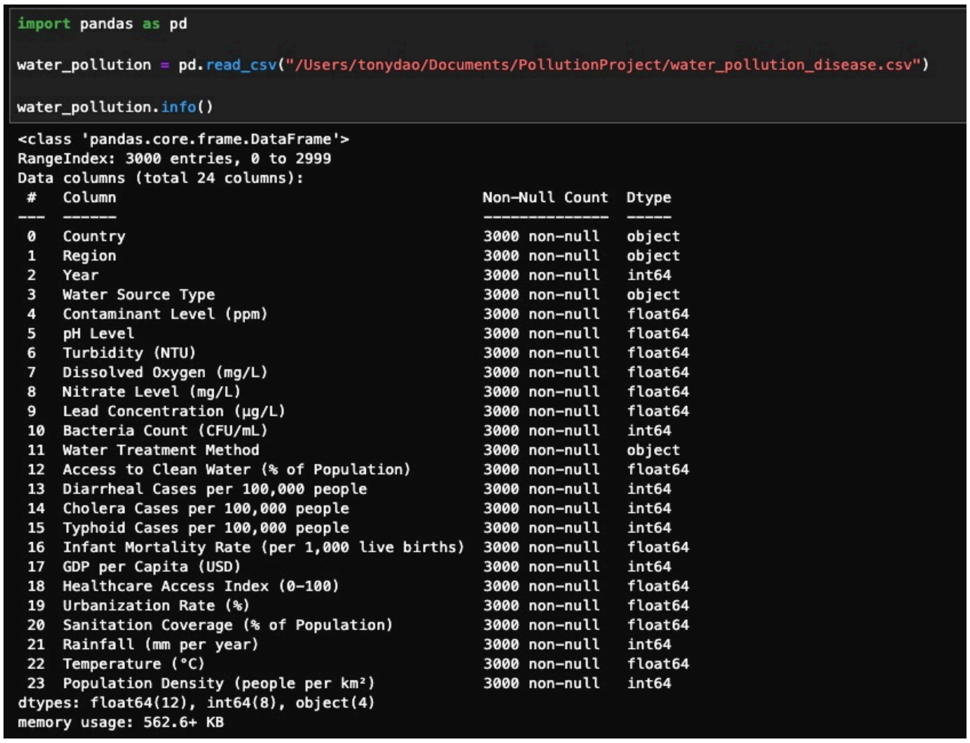
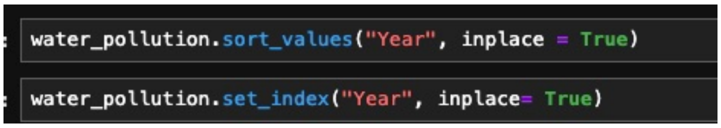
1. I first import my data file using the whole path name.
2. I then use the info.() to see what types of data type im working with. This also gives me my columns.



```
import pandas as pd

water_pollution = pd.read_csv("/Users/tonydao/Documents/PollutionProject/water_pollution_disease.csv")

water_pollution.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 24 columns):
 #   Column                                   Non-Null Count  Dtype
---  ------                                   --------------  -----
 0   Country                                  3000 non-null   object
 1   Region                                   3000 non-null   object
 2   Year                                     3000 non-null   int64
 3   Water Source Type                        3000 non-null   object
 4   Contaminant Level (ppm)                  3000 non-null   float64
 5   pH Level                                 3000 non-null   float64
 6   Turbidity (NTU)                          3000 non-null   float64
 7   Dissolved Oxygen (mg/L)                  3000 non-null   float64
 8   Nitrate Level (mg/L)                     3000 non-null   float64
 9   Lead Concentration (µg/L)                3000 non-null   float64
 10  Bacteria Count (CFU/mL)                  3000 non-null   int64
 11  Water Treatment Method                   3000 non-null   object
 12  Access to Clean Water (% of Population)  3000 non-null   float64
 13  Diarrheal Cases per 100,000 people       3000 non-null   int64
 14  Cholera Cases per 100,000 people         3000 non-null   int64
 15  Typhoid Cases per 100,000 people         3000 non-null   int64
 16  Infant Mortality Rate (per 1,000 live births) 3000 non-null   float64
 17  GDP per Capita (USD)                     3000 non-null   int64
 18  Healthcare Access Index (0-100)          3000 non-null   float64
 19  Urbanization Rate (%)                    3000 non-null   float64
 20  Sanitation Coverage (% of Population)    3000 non-null   float64
 21  Rainfall (mm per year)                   3000 non-null   int64
 22  Temperature (°C)                         3000 non-null   float64
 23  Population Density (people per km²)      3000 non-null   int64
dtypes: float64(12), int64(8), object(4)
memory usage: 562.6+ KB
```

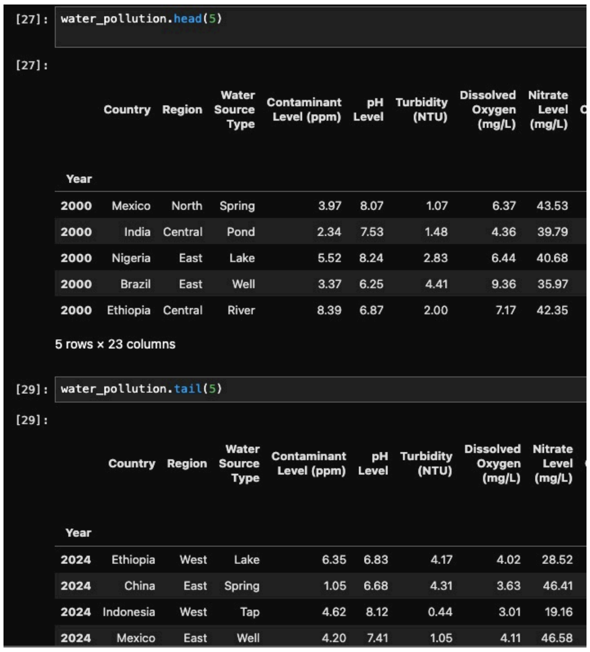
3. I notice that Year is a column choice that would be good to start out as a sorted index. This would allow me to conduct basic chart/graph observation when comparing it with other column variables.



```
water_pollution.sort_values("Year", inplace = True)
```

```
water_pollution.set_index("Year", inplace= True)
```

4: I used the .head() and .tail() to see if my index has been changed and if it is sorted corrected.



[27]:
```
water_pollution.head(5)
```

[27]:

Year	Country	Region	Water Source Type	Contaminant Level (ppm)	pH Level	Turbidity (NTU)	Dissolved Oxygen (mg/L)	Nitrate Level (mg/L)
2000	Mexico	North	Spring	3.97	8.07	1.07	6.37	43.53
2000	India	Central	Pond	2.34	7.53	1.48	4.36	39.79
2000	Nigeria	East	Lake	5.52	8.24	2.83	6.44	40.68
2000	Brazil	East	Well	3.37	6.25	4.41	9.36	35.97
2000	Ethiopia	Central	River	8.39	6.87	2.00	7.17	42.35

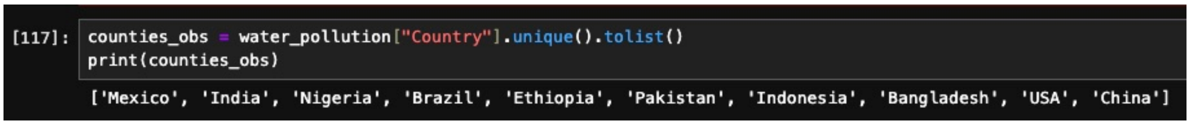
5 rows × 23 columns

[29]:
```
water_pollution.tail(5)
```

[29]:

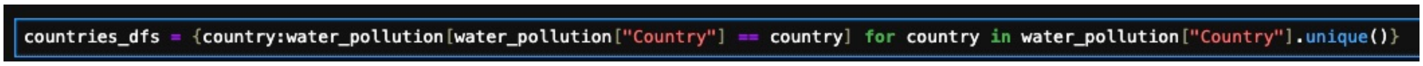
Year	Country	Region	Water Source Type	Contaminant Level (ppm)	pH Level	Turbidity (NTU)	Dissolved Oxygen (mg/L)	Nitrate Level (mg/L)
2024	Ethiopia	West	Lake	6.35	6.83	4.17	4.02	28.52
2024	China	East	Spring	1.05	6.68	4.31	3.63	46.41
2024	Indonesia	West	Tap	4.62	8.12	0.44	3.01	19.16
2024	Mexico	East	Well	4.20	7.41	1.05	4.11	46.58

5: Going back to our dataset, we can see the Country column. We can create sub-dataframe for each unique country. I first wanted to see what are my unique strings in the Country column. Then I will convert this to a list.

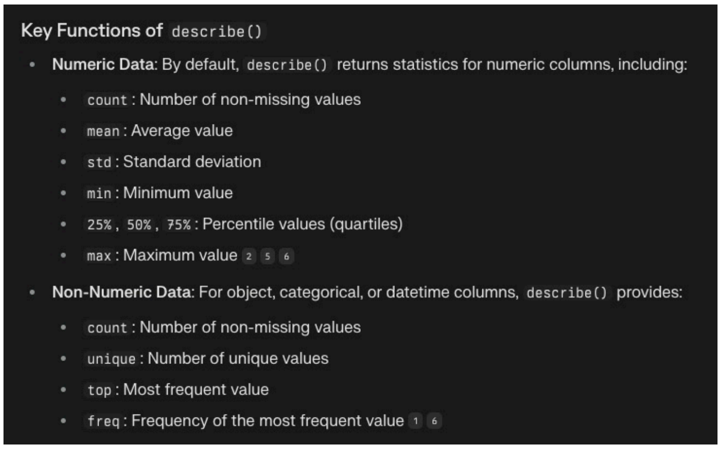


```
[117]: counties_obs = water_pollution["Country"].unique().tolist()
       print(counties_obs)

       ['Mexico', 'India', 'Nigeria', 'Brazil', 'Ethiopia', 'Pakistan', 'Indonesia', 'Bangladesh', 'USA', 'China']
```

7: We can subdivide the water_pollution DF into their respective countries and analyze each one one separately. Since we have 10 different countries, we opt to use a for loop that takes our Country col in water_pollution using the .unique() method. For each 'country' in that list, the code select where "Country" equals that 'country'. The key will be the country's name and its value will be its dataframe.



```
countries_dfs = {country:water_pollution[water_pollution["Country"] == country] for country in water_pollution["Country"].unique()}
```

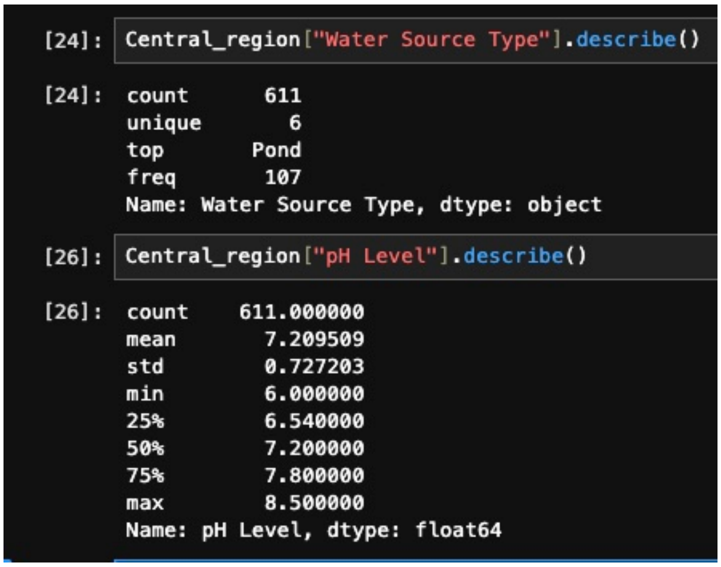


Key Functions of `describe()`

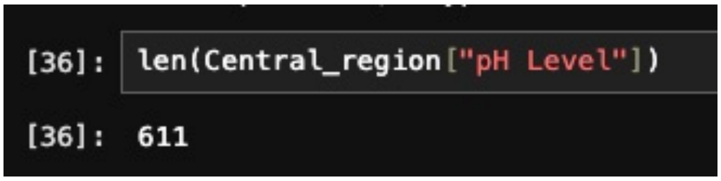
- **Numeric Data**: By default, `describe()` returns statistics for numeric columns, including:
  - `count` : Number of non-missing values
  - `mean` : Average value
  - `std` : Standard deviation
  - `min` : Minimum value
  - `25%` , `50%` , `75%` : Percentile values (quartiles)
  - `max` : Maximum value [2] [5] [6]
- **Non-Numeric Data**: For object, categorical, or datetime columns, `describe()` provides:
  - `count` : Number of non-missing values
  - `unique` : Number of unique values
  - `top` : Most frequent value
  - `freq` : Frequency of the most frequent value [1] [6]

8: We can use the describe() to generate descriptive statistics that summarizes the central tendency, dispersion, and shape of a dataset's distribution, excluding NaN values.

8a: We can see the difference in output when applying it to our "pH Level" and "Water Source Type" col.

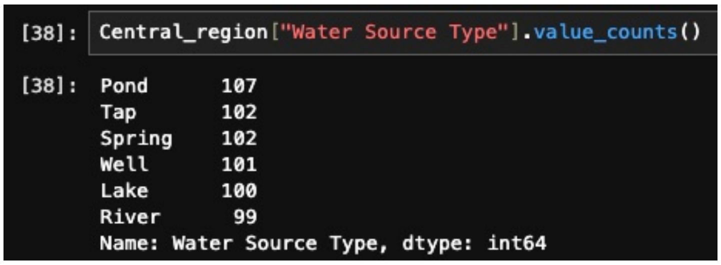


```
[24]: Central_region["Water Source Type"].describe()

[24]: count      611
      unique       6
      top       Pond
      freq       107
      Name: Water Source Type, dtype: object

[26]: Central_region["pH Level"].describe()

[26]: count    611.000000
      mean       7.209509
      std        0.727203
      min        6.000000
      25%        6.540000
      50%        7.200000
      75%        7.800000
      max        8.500000
      Name: pH Level, dtype: float64
```



```
[36]: len(Central_region["pH Level"])

[36]: 611
```

8b: By applying the len(), we can see how many rows are in that col.



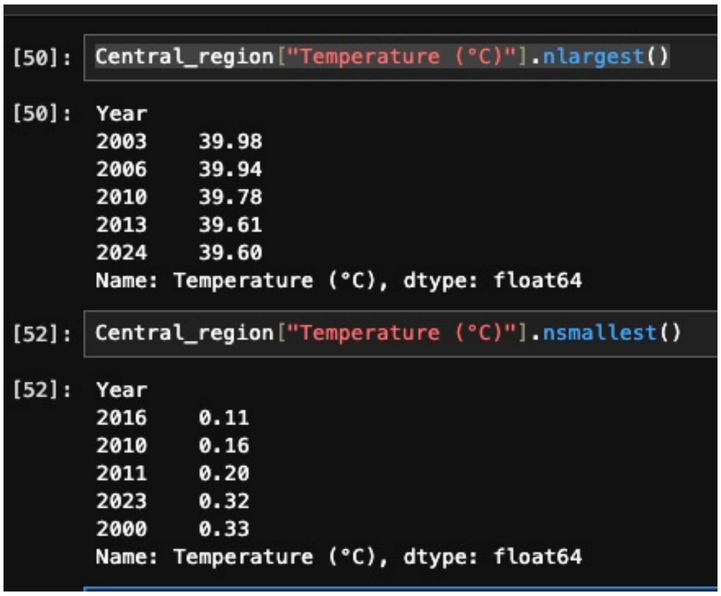
```
[38]: Central_region["Water Source Type"].value_counts()

[38]: Pond      107
      Tap       102
      Spring    102
      Well      101
      Lake      100
      River      99
      Name: Water Source Type, dtype: int64
```

8c: We can also calculate
the .median(), .std(), .min() and max().
8d: .nunique() returns the number of unique values
in a Series or DF along a specified axis.
The .unique returns a NumPy array of the unique
values, in the order they appear.

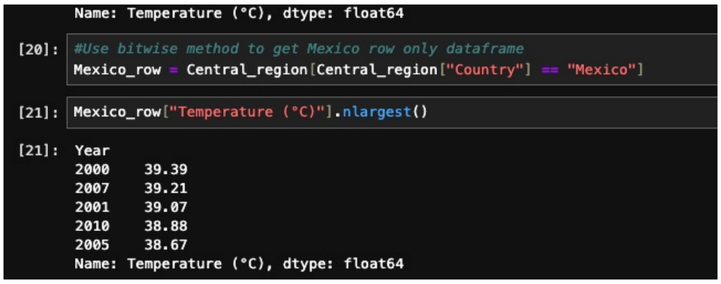
9: use the value_counts() method in pandas when you want to count the frequency of unique values in a Series (such as a DataFrame column). Here we applied the method to see the frequency number of the "Water Source Type" col in the Central_region DF.

9a: The "normalized" parameter gives us the relative frequency. We can also order the count numbers by using the "ascending" parameter.

10: The nlargest() and nsmallest() method gives us the top n rows of the highest or lowest values. These methods are optimized for speed and memory efficiency. This is a partial sorting algorithms. The default is 5 rows.

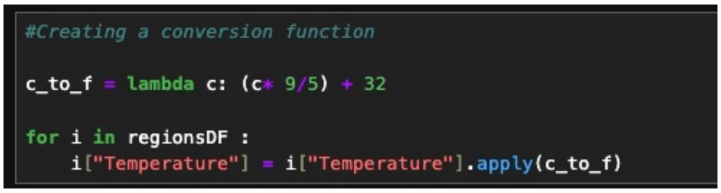


```
[50]: Central_region["Temperature (°C)"].nlargest()

[50]: Year
      2003    39.98
      2006    39.94
      2010    39.78
      2013    39.61
      2024    39.60
      Name: Temperature (°C), dtype: float64

[52]: Central_region["Temperature (°C)"].nsmallest()

[52]: Year
      2016    0.11
      2010    0.16
      2011    0.20
      2023    0.32
      2000    0.33
      Name: Temperature (°C), dtype: float64
```

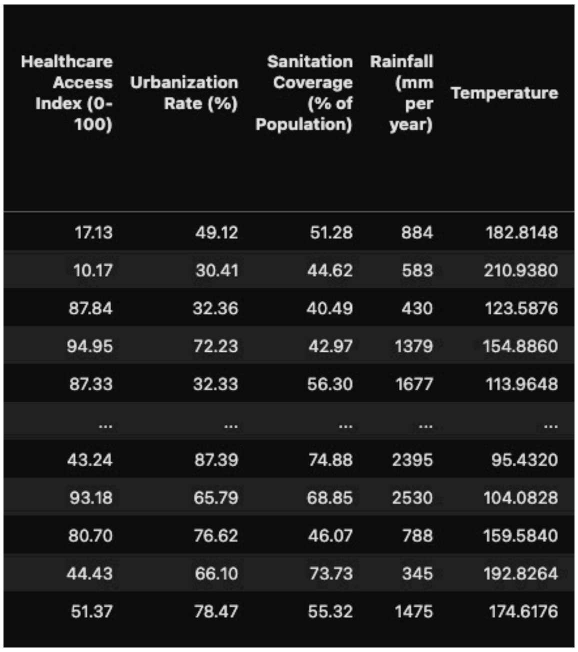


```
      Name: Temperature (°C), dtype: float64

[20]: #Use bitwise method to get Mexico row only dataframe
      Mexico_row = Central_region[Central_region["Country"] == "Mexico"]

[21]: Mexico_row["Temperature (°C)"].nlargest()

[21]: Year
      2000    39.39
      2007    39.21
      2001    39.07
      2010    38.88
      2005    38.67
      Name: Temperature (°C), dtype: float64
```

11: We created a Mexico only dataframe and looked at the highest 5 recorded temperature.

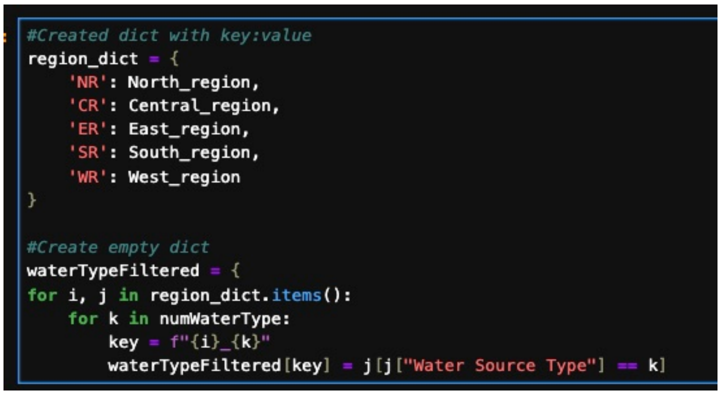


```
#Creating a conversion function

c_to_f = lambda c: (c* 9/5) + 32

for i in regionsDF :
    i["Temperature"] = i["Temperature"].apply(c_to_f)
```



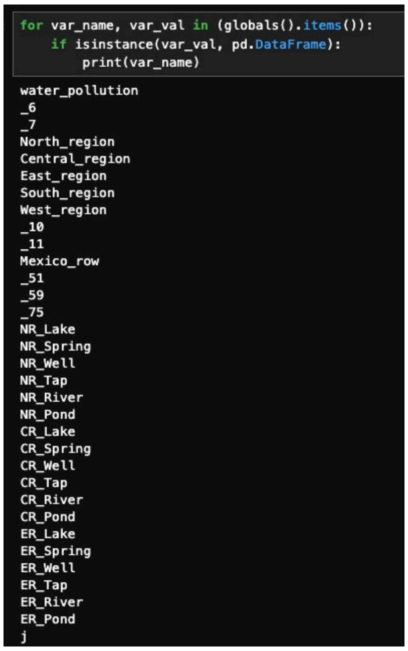
Healthcare Access Index (0-100)	Urbanization Rate (%)	Sanitation Coverage (% of Population)	Rainfall (mm per year)	Temperature
17.13	49.12	51.28	884	182.8148
10.17	30.41	44.62	583	210.9380
87.84	32.36	40.49	430	123.5876
94.95	72.23	42.97	1379	154.8860
87.33	32.33	56.30	1677	113.9648
...	...	...	...	...
43.24	87.39	74.88	2395	95.4320
93.18	65.79	68.85	2530	104.0828
80.70	76.62	46.07	788	159.5840
44.43	66.10	73.73	345	192.8264
51.37	78.47	55.32	1475	174.6176

12: We can convert our temperature cols unit from C to F.



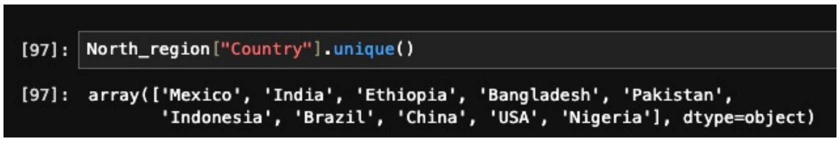
```
#Created dict with key:value
region_dict = {
    'NR': North_region,
    'CR': Central_region,
    'ER': East_region,
    'SR': South_region,
    'WR': West_region
}

#Create empty dict
waterTypeFiltered = {
for i, j in region_dict.items():
    for k in numWaterType:
        key = f"{i}_{k}"
        waterTypeFiltered[key] = j[j["Water Source Type"] == k]
```

13: We can now create sub DF based on water source types



```
for var_name, var_val in (globals().items()):
    if isinstance(var_val, pd.DataFrame):
        print(var_name)
```

```
water_pollution
_6
_7
North_region
Central_region
East_region
South_region
West_region
_10
_11
Mexico_row
_51
_59
_75
NR_Lake
NR_Spring
NR_Well
NR_Tap
NR_River
NR_Pond
CR_Lake
CR_Spring
CR_Well
CR_Tap
CR_River
CR_Pond
ER_Lake
ER_Spring
ER_Well
ER_Tap
ER_River
ER_Pond
j
```

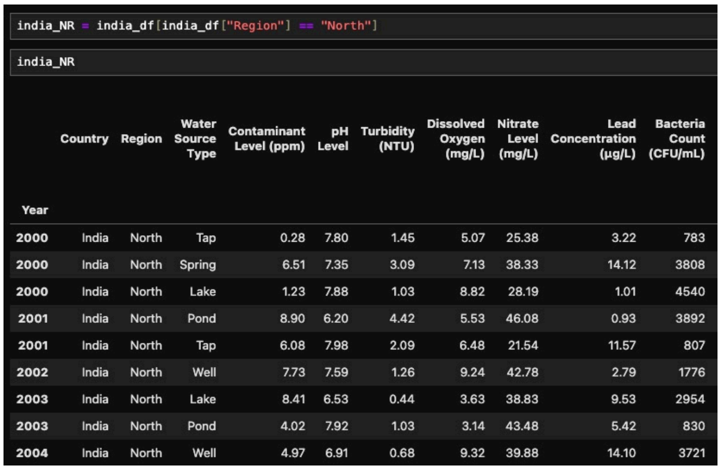


```
[97]: North_region["Country"].unique()

[97]: array(['Mexico', 'India', 'Ethiopia', 'Bangladesh', 'Pakistan',
             'Indonesia', 'Brazil', 'China', 'USA', 'Nigeria'], dtype=object)
```

14: Now we can do basic indepth investigation on each region. The first region of focus will be the central region. I used the unique() on the

ANALYSIS OF INDIA

1: We will create a northern India region along with other regions.



```
india_NR = india_df[india_df["Region"] == "North"]
```

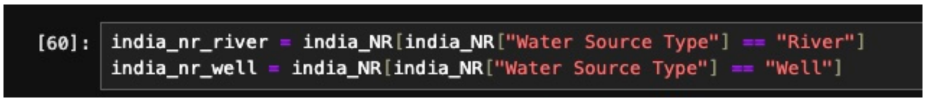
```
india_NR
```

	Country	Region	Water Source Type	Contaminant Level (ppm)	pH Level	Turbidity (NTU)	Dissolved Oxygen (mg/L)	Nitrate Level (mg/L)	Lead Concentration (µg/L)	Bacteria Count (CFU/mL)
Year										
2000	India	North	Tap	0.28	7.80	1.45	5.07	25.38	3.22	783
2000	India	North	Spring	6.51	7.35	3.09	7.13	38.33	14.12	3808
2000	India	North	Lake	1.23	7.88	1.03	8.82	28.19	1.01	4540
2001	India	North	Pond	8.90	6.20	4.42	5.53	46.08	0.93	3892
2001	India	North	Tap	6.08	7.98	2.09	6.48	21.54	11.57	807
2002	India	North	Well	7.73	7.59	1.26	9.24	42.78	2.79	1776
2003	India	North	Lake	8.41	6.53	0.44	3.63	38.83	9.53	2954
2003	India	North	Pond	4.02	7.92	1.03	3.14	43.48	5.42	830
2004	India	North	Well	4.97	6.91	0.68	9.32	39.88	14.10	3721

- From the options of water source type, Northern India contains an abundance of rivers and groundwater wells.

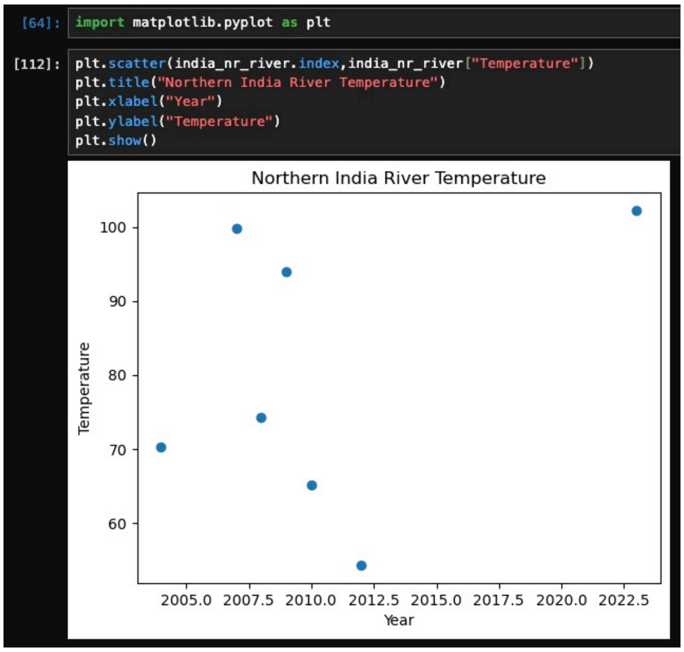
WaterAid. (n.d.). Groundwater quality information North India. WASH Matters. https://washmatters.wateraid.org/publications/groundwater-quality-information-north-india

Based on these information we will focus on these two water source type

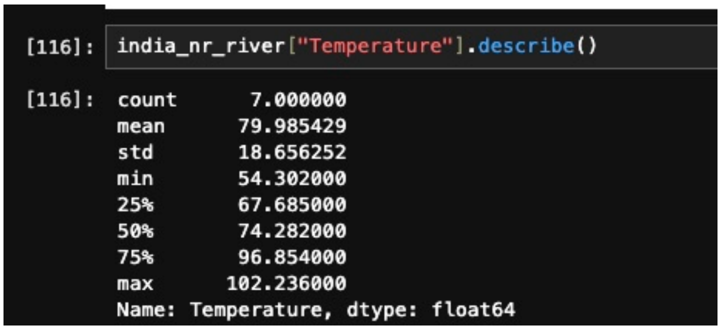


```
[60]: india_nr_river = india_NR[india_NR["Water Source Type"] == "River"]
      india_nr_well = india_NR[india_NR["Water Source Type"] == "Well"]
```

- The first thing I did was observe to see if there is anything meaningful for temperature vs year



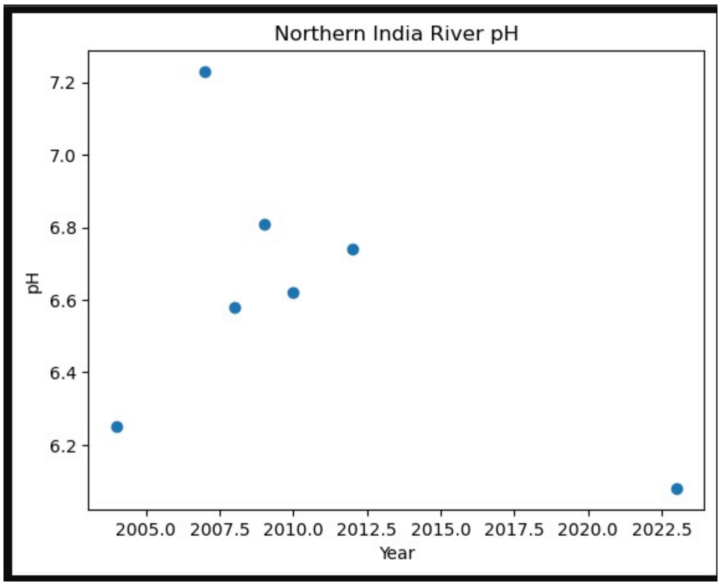
```
[64]: import matplotlib.pyplot as plt

[112]: plt.scatter(india_nr_river.index,india_nr_river["Temperature"])
       plt.title("Northern India River Temperature")
       plt.xlabel("Year")
       plt.ylabel("Temperature")
       plt.show()
```

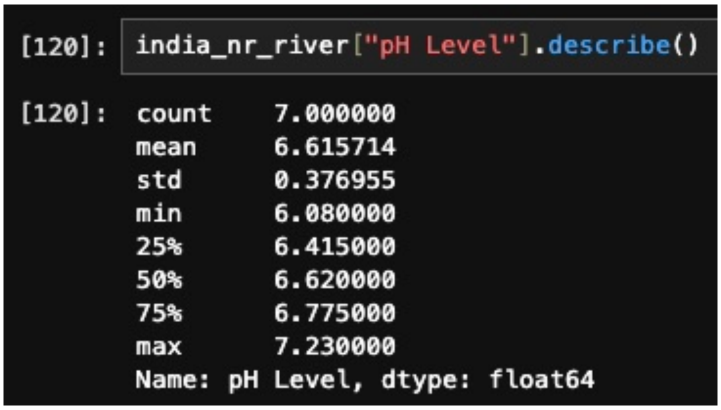


```
[116]: india_nr_river["Temperature"].describe()

[116]: count      7.000000
       mean      79.985429
       std       18.656252
       min       54.302000
       25%       67.685000
       50%       74.282000
       75%       96.854000
       max      102.236000
       Name: Temperature, dtype: float64
```

  - Nothing meaningful was extracted from this graph. Temperature fluctuates with the highest temperature recorded was 102.2F in 2023 and the lowest being 54.3F in 2012.

- The next thing we can observe would be the pH level vs Year





```
[120]: india_nr_river["pH Level"].describe()

[120]: count    7.000000
       mean     6.615714
       std      0.376955
       min      6.080000
       25%      6.415000
       50%      6.620000
       75%      6.775000
       max      7.230000
       Name: pH Level, dtype: float64
```

  - From this graph it is difficult to extract any information. The basic stat analysis tells us that the lowest pH recorded was in 2023, which was 6.0. The highest recorded pH was 7.2 on 2007

- It may be difficult to extract anything meaningful due to the numbers of rows. We can combine both the river and well since they are both freshwater.