



HEI Research Report 233

Hydrocarbon Extraction and Risk to Groundwater in Pennsylvania: Part 2. An Analysis of Water Quality in the Oldest Hydrocarbon Basin in the United States

Jennifer Baka et al.

Additional Materials 5. Geochemical Statistics Code (May 29)

Correspondence may be addressed to Dr. Jennifer Baka, Department of Geography, 302 Walker Building, University Park, PA 16802; email: jeb525@psu.edu.

Although this report was produced with partial funding by the United States Environmental Protection Agency under Contract No. 68HERC19D0010 to the Health Effects Institute, it has not been subjected to the Agency's peer and administrative review and may not reflect the views of the Agency; thus, no official endorsement by the Agency should be inferred. This report also has not been reviewed by private party institutions, including those that support HEI Energy, and may not reflect the views or policies of these parties; thus, no endorsement by them should be inferred.

© 2026 Health Effects Institute, One Beacon Street, Suite 21300, Boston, MA 02108

Additional Materials 5. Geochemical Statistics Code (May 29)

Data import & preprocessing

```
In [88]: # Import csv file from "HEI/Finalized_results" folder
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os, sys
```

```
In [89]: # import radium data
from pathlib import Path
root = Path.cwd().parent

DATA_DIR = root/'Dataset'
OUTPUT_DIR = root/'Results'

df_radium = pd.read_excel(DATA_DIR/'radium_data.xlsx', header=0)
df_radium.head()
```

```
Out[89]:
```

	Sample ID	Ra - 226 (pCi/L)	Ra - 228 (pCi/L)	Ra 226 + 228 (pCi/L)	228Ra/226Ra
0	GR_Gilmore_093	0.231318	0.265451	0.496769	1.147559
1	GR_Gilmore_094	0.054557	0.122037	0.176594	2.236875
2	WA_Amwell_026	0.257883	0.441788	0.699672	1.713132
3	WA_Amwell_027	0.033657	0.054951	0.088608	1.632678
4	WA_Amwell_028	0.036988	0.060096	0.097084	1.624743

```
In [90]: df_radium['sample number'] = df_radium['Sample ID'].str.split('_').str[-1]
df_radium.head()
```

Out[90]:

	Sample ID	Ra - 226 (pCi/L)	Ra - 228 (pCi/L)	Ra 226 + 228 (pCi/L)	228Ra/226Ra	sample number
0	GR_Gilmore_093	0.231318	0.265451	0.496769	1.147559	093
1	GR_Gilmore_094	0.054557	0.122037	0.176594	2.236875	094
2	WA_Amwell_026	0.257883	0.441788	0.699672	1.713132	026
3	WA_Amwell_027	0.033657	0.054951	0.088608	1.632678	027
4	WA_Amwell_028	0.036988	0.060096	0.097084	1.624743	028

In [91]:

```
# Change the name of some columns
df_radium.rename(columns={'Ra - 226 (pCi/L)': 'Ra-226',
                          'Ra - 228 (pCi/L)': 'Ra-228',
                          'Ra 226 + 228 (pCi/L)': 'Ra 226 + 228'}, inplace=True)
```

In [92]:

```
# Import csv file
# Each csv file contains:
### df_final: all samples (without stream samples) with analytes' concentrations and features for stats/spatial analysis
### df_full: all samples with analytes' concentrations only (used for easier set up of stats comparison)
### df_unit: unit of analytes
### df_epa: EPA MCL/SMCL standards and thresholds related to potential health risks
df_final = pd.read_csv(DATA_DIR/'df_final.csv', header = 0)
df_full = pd.read_csv(DATA_DIR/'Full_Table_Data_for_stats.csv', header = 0)
df_unit = pd.read_csv(DATA_DIR/'Full_Table_Units.csv', header = 0)
df_epa = pd.read_csv(DATA_DIR/'EPA_Standard_with_health_risk.csv', header = 0)
# df_fix = pd.read_excel('D:/My Drive/HEI/fixed_effect.xlsx', header = 0, na_values=-1)
```

In [116...]

```
# Find out range of Cl/Br ratio in df_full
df_full['Cl/Br'] = df_full['Cl'] / df_full['Br']
print(f"Range of Cl/Br ratio: {df_full['Cl/Br'].min()} to {df_full['Cl/Br'].max()}")

# how many samples have ratio >300 ?
high_cl_br_count = (df_full['Cl/Br'] > 300).sum()
print(f"Number of samples with Cl/Br ratio > 300: {high_cl_br_count}")

# how many samples have ratio < 100 ?
low_cl_br_count = (df_full['Cl/Br'] < 100).sum()
print(f"Number of samples with Cl/Br ratio < 100: {low_cl_br_count}")
```

```
# how many samples have ratio between 100 and 300
mid_cl_br_count = ((df_full['Cl/Br'] >= 100) & (df_full['Cl/Br'] <= 300)).sum()
print(f"Number of samples with Cl/Br ratio between 100 and 300: {mid_cl_br_count}")
```

Range of Cl/Br ratio: 29.15255172413798 to 4721.298701298704

Number of samples with Cl/Br ratio > 300: 52

Number of samples with Cl/Br ratio < 100: 6

Number of samples with Cl/Br ratio between 100 and 300: 32

In [93]: `df_final.columns`

```
Out[93]: Index(['Site ID', 'Address', 'X', 'Y', 'Sample Group', 'Date', 'Time Sampled',
              'Source', 'Temp', 'DO', 'SC', 'Field pH', 'Lab pH', 'Alkalinity',
              'Titration Date', 'Gas Rover Reading', 'F', 'Cl', 'SO4', 'Br', 'NO3',
              'Al', 'Ba', 'Ca', 'Fe', 'K', 'Mg', 'Mn', 'Na', 'P', 'Si', 'Sr', 'Ti',
              'Li', 'Be', 'Ni', 'Cu', 'As', 'Sr.1', 'Cd', 'Tl', 'Pb', 'U',
              '87Sr/86Sr', '87Sr/86Sr Error', '87Sr/86Sr Ratio', 'Benzene', 'Toluene',
              'Ethylbenzene', 'm,p-Xylene', 'o-Xylene', 'Methane', 'Ethane',
              'Propane', 'δ13C1(‰)', 'δDC1(‰)', 'δ13C2(‰)', 'DIC (mg C/L)',
              'DIC (SD)', 'DIC (mol/L C)', '[H+]', '[OH-]', 'a0 [H2CO3]',
              'a1 [HCO3-]', 'a2 [CO32-]', 'Alk (eq/L)', 'Alk (mg CaCO3/L)', 'Radon',
              'Type', 'NEAR_UOG_DIST', 'Impoundment_DIST', 'Recent_Spill_DIST'],
              dtype='object')
```

```
In [94]: # caculate the number of samples from sample group in df_final
df_final['Sample Group'].value_counts()
```

```
Out[94]: Sample Group
Other      27
HS         17
I          14
S          14
HS/I       10
PureC       8
HS/S        3
Name: count, dtype: int64
```

```
In [95]: ### Remove the AMD and rain samples
### Drop rows that has Site ID equals to Wa_Wbeth_022, Wa_WBeth_029, GR_Whtly_091
df_final = df_final.loc[~df_final['Site ID'].isin(['WA_Wbeth_022', 'WA_WBeth_029', 'GR_Whtly_091']),:]
df_full = df_full.loc[~df_full['Site ID'].isin(['WA_Wbeth_022', 'WA_WBeth_029', 'GR_Whtly_091']),:]
df_final.shape
```

Out[95]: (90, 72)

```
In [96]: df_unit = df_unit.set_index('Unnamed: 0')
df_unit.index.name = None
df_unit
```

Out[96]:

	Unit
Site ID	-
Sample Group	-
Date	-
Time Sampled	-
Source	-
...	...
228Ra/226Ra	pCi/L
Ra 226+228	pCi/L
Ra226	pCi/L
Ra228	pCi/L
Xylene(total)	µg/L

79 rows × 1 columns

```
In [97]: # Add two rows for Radium-226 and Radium-228 units
df_unit.loc['Ra-226'] = 'pCi/L'
df_unit.loc['Ra-228'] = 'pCi/L'
df_unit.loc['228Ra/226Ra'] = '-'
df_unit
```

Out[97]:

	Unit
Site ID	-
Sample Group	-
Date	-
Time Sampled	-
Source	-
...	...
Ra226	pCi/L
Ra228	pCi/L
Xylene(total)	µg/L
Ra-226	pCi/L
Ra-228	pCi/L

81 rows × 1 columns

In [98]:

```
df_epa
```

Out[98]:

	list	MCL/SMCL limit
0	DO	-
1	SC	-
2	Field pH	6.5-8.5
3	Lab pH	6.5-8.5
4	Alkalinity	-
5	Gas Rover Reading	-
6	F	2
7	Cl	250
8	SO4	250
9	Br	2
10	NO3	44
11	Al	0.05-0.2
12	Ba	2
13	Ca	-
14	Fe	0.3
15	K	-
16	Mg	-
17	Mn	0.05
18	Na	-
19	P	-
20	Si	-
21	Sr	4

	list	MCL/SMCL limit
22	Li	10
23	Be	4
24	Ni	100
25	Cu	1000
26	As	10
27	Cd	5
28	Tl	2
29	Pb	15
30	U	30
31	87Sr/86Sr	-
32	87Sr/86Sr Ratio	-
33	Benzene	5
34	Toluene	1000
35	Ethylbenzene	700
36	m,p-Xylene	10000
37	o-Xylene	10000
38	Xylene(total)	10000
39	Methane	10
40	Ethane	-
41	Propane	-
42	$\delta^{13}C1(\text{‰})$	-
43	$\delta DC1(\text{‰})$	-

	list	MCL/SMCL limit
44	$\delta^{13}\text{C}_2(\text{‰})$	-
45	DIC (mg C/L)	-
46	DIC (SD)	-
47	DIC (mol/L C)	-
48	[H+]	-
49	[OH-]	-
50	a0 [H ₂ CO ₃]	-
51	a1 [HCO ₃ ⁻]	-
52	a2 [CO ₃ ²⁻]	-
53	Alk (eq/L)	-
54	Alk (mg CaCO ₃ /L)	-
55	Radon	300

```
In [99]: # Add rows of threshold values for Radium-226 and Radium-228
df_epa_copy = df_epa.copy()
new_rows = pd.DataFrame({'list':['Ra-226', 'Ra-228', 'Ra 226 + 228', '228Ra/226Ra'],
                        'MCL/SMCL limit':[27,2.7,5,'-']})
df_epa_copy = pd.concat([df_epa_copy, new_rows], ignore_index=True)
df_epa_copy
```

Out[99]:

	list	MCL/SMCL limit
0	DO	-
1	SC	-
2	Field pH	6.5-8.5
3	Lab pH	6.5-8.5
4	Alkalinity	-
5	Gas Rover Reading	-
6	F	2
7	Cl	250
8	SO4	250
9	Br	2
10	NO3	44
11	Al	0.05-0.2
12	Ba	2
13	Ca	-
14	Fe	0.3
15	K	-
16	Mg	-
17	Mn	0.05
18	Na	-
19	P	-
20	Si	-
21	Sr	4

	list	MCL/SMCL limit
22	Li	10
23	Be	4
24	Ni	100
25	Cu	1000
26	As	10
27	Cd	5
28	Tl	2
29	Pb	15
30	U	30
31	87Sr/86Sr	-
32	87Sr/86Sr Ratio	-
33	Benzene	5
34	Toluene	1000
35	Ethylbenzene	700
36	m,p-Xylene	10000
37	o-Xylene	10000
38	Xylene(total)	10000
39	Methane	10
40	Ethane	-
41	Propane	-
42	$\delta^{13}C1(\text{‰})$	-
43	$\delta^{13}C1(\text{‰})$	-

	list	MCL/SMCL limit
44	$\delta^{13}\text{C}_2(\text{‰})$	-
45	DIC (mg C/L)	-
46	DIC (SD)	-
47	DIC (mol/L C)	-
48	[H+]	-
49	[OH-]	-
50	a0 [H ₂ CO ₃]	-
51	a1 [HCO ₃ ⁻]	-
52	a2 [CO ₃ ²⁻]	-
53	Alk (eq/L)	-
54	Alk (mg CaCO ₃ /L)	-
55	Radon	300
56	Ra-226	27
57	Ra-228	2.7
58	Ra 226 + 228	5
59	228Ra/226Ra	-

In [100... `df_epa = df_epa_copy`

In [101... `# Adjust the radium sample numbers to match with df_final`
`match_index = df_radium[df_radium['sample number'].isin(['013 - 2', '013 - 3'])].index`
`df_radium.loc[match_index, 'sample number'] = ['013-2', '013-3']`

In [102... `# Merge radium data into df_full`
`df_full['sample number'] = df_full['Site ID'].str.split('_').str[-1]`

```
df_full = pd.merge(df_full, df_radium[['sample number', 'Ra-226', 'Ra-228', 'Ra 226 + 228', '228Ra/226Ra']], on='sample number',
df_full.columns
```

```
Out[102...] Index(['Unnamed: 0', 'Site ID', 'DO', 'SC', 'Field pH', 'Lab pH', 'Alkalinity',
      'Gas Rover Reading', 'F', 'Cl', 'SO4', 'Br', 'NO3', 'Al', 'Ba', 'Ca',
      'Fe', 'K', 'Mg', 'Mn', 'Na', 'P', 'Si', 'Sr', 'Ti', 'Li', 'Be', 'Ni',
      'Cu', 'As', 'Cd', 'Tl', 'Pb', 'U', '87Sr/86Sr', 'Benzene', 'Toluene',
      'Ethylbenzene', 'm,p-Xylene', 'o-Xylene', 'Xylene(total)', 'Methane',
      'Ethane', 'Propane', 'δ13C1(‰)', 'δDC1(‰)', 'δ13C2(‰)', 'Radon',
      'sample number', 'Ra-226', 'Ra-228', 'Ra 226 + 228', '228Ra/226Ra'],
      dtype='object')
```

```
In [103...] df_full.drop(columns=['sample number'], inplace=True)
```

```
In [104...] # After Cell 15 – check for row inflation
print("df_full shape:", df_full.shape)
print("Duplicate sample numbers in df_radium:")
print(df_radium['sample number'].value_counts()[lambda x: x > 1])
```

df_full shape: (90, 52)

Duplicate sample numbers in df_radium:

Series([], Name: count, dtype: int64)

Define functions that generate stats summary

- **epa_table(df):**
 - 'list' of analytes
 - 'EPA Standard' regulated thresholds
 - '% exceeding EPA Standard' among all samples
- **stats_summary(df,df_unit,epa):**
 - unit & number of valid measurement
 - minimum, maximum, mean, median, and 5% and 95% percentiles
 - epa table

```
In [105...] # Define a function that generate epa tables
def epa_table(df):
```

```

epa = df_epa
# Create a new column that stores the percentage values in df
epa['Percentage'] = "-"
for col in df.columns:
    # EPA secondary standard for pH : 6.5 - 8.5
    if 'pH' in col:
        sum_pH = ((df[col] < 6.5) | (df[col] > 8.5)).sum()
        perc = f"{sum_pH / df[col].count() * 100:.1f}% ({sum_pH})"
        epa.loc[epa['list']== col, 'Percentage'] = perc

    # EPA secondary standard for Al : 0.05 - 0.2 mg/L, here would be using 0.2 mg/L as the threshold
    elif col == 'Al' :
        sum_al = (df[col] > 0.2).sum()
        perc = f"{sum_al / df[col].count() * 100:.1f}% ({sum_al})"
        epa.loc[epa['list']== col, 'Percentage'] = perc

    # Other EPA standards according to the table
    elif epa.loc[epa['list']== col, 'MCL/SMCL limit'].astype(str).ne('-').any():
        epa_standard = epa.loc[epa['list'] == col, 'MCL/SMCL limit'].astype(float).iloc[0]
        sum = (df[col] > epa_standard).sum()
        perc = f"{sum / df[col].count() * 100:.1f}% ({sum})"
        epa.loc[epa['list']== col, 'Percentage'] = perc
epa = epa.rename(columns={'Percentage': '% Exceeding MCL/SMCL'})
return epa

```

In [106... *# Define a function that generate stats summary*

```

def stats_summary(df,df_unit,epa):
    num_values = df.count()
    min_values = df.min()
    max_values = df.max()
    mean_values = df.mean()
    median_values = df.median(axis=0)
    upper_quartile = df.quantile(0.75, axis=0)
    lower_quartile = df.quantile(0.25, axis=0)
    upper_95_percentile = df.quantile(0.95, axis=0)
    lower_5_percentile = df.quantile(0.05, axis=0)

    df_summary = pd.DataFrame({
        'n' : num_values,
        'Min': min_values,
        'Max': max_values,

```

```

'Mean': mean_values,
'Median': median_values,
'5th Percentile': lower_5_percentile,
'25th Percentile': lower_quartile,
'75th Percentile': upper_quartile,
'95th Percentile': upper_95_percentile
})

# Remove row with index 'Ti'
df_summary = df_summary.drop(index='Ti')

# Keep 3 digits after decimal for all values other than Sr ratio. This may exclude some values that are too small
df_summary.loc[df_summary.index != '87Sr/86Sr'] = df_summary.loc[df_summary.index != '87Sr/86Sr'].round(3)
df_summary.loc[df_summary.index == '87Sr/86Sr'] = df_summary.loc[df_summary.index == '87Sr/86Sr'].round(4)

# Append the units from df_unit and EPA standards & percentages from df_epa
df_summary_1 = pd.merge(df_summary, df_unit, left_index=True, right_index=True, how='left')
df_summary_2 = pd.merge(df_summary_1, epa, left_on=df_summary_1.index, right_on='list', how='left')
df_summary_2 = df_summary_2.set_index(df_summary_1.index)
df_summary_2 = df_summary_2.drop(columns=['list'])

# Fill na with "-"
df_summary_2 = df_summary_2.fillna('-')

# Move the units to the first column
units = df_summary_2.pop('Unit')
df_summary_2.insert(0, 'Unit', units)

# Make the index as the first column
df_summary_2 = df_summary_2.reset_index()
df_summary_2 = df_summary_2.rename(columns={'index': 'Analyte'})

# Delete the line where analyte is 'Lab pH'
df_summary_2 = df_summary_2.loc[df_summary_2['Analyte'] != 'Lab pH', :]

# Change the name of analyte 'Field pH' to 'pH', and 'Gas Rover Reading' to 'Air [CH4]'
df_summary_2['Analyte'] = df_summary_2['Analyte'].replace('Field pH', 'pH')
df_summary_2['Analyte'] = df_summary_2['Analyte'].replace('Gas Rover Reading', 'Air [CH4]')
df_summary_2['Analyte'] = df_summary_2['Analyte'].replace('NO3', 'NO3-NO3')

```

```
return df_summary_2
```

Full table stats summary: All groundwater and spring samples (Table 18)

```
In [107... df_full.shape
```

```
Out[107... (90, 52)
```

```
In [108... df_final.shape
```

```
Out[108... (90, 72)
```

```
In [109... df_full.columns
```

```
Out[109... Index(['Unnamed: 0', 'Site ID', 'DO', 'SC', 'Field pH', 'Lab pH', 'Alkalinity',
      'Gas Rover Reading', 'F', 'Cl', 'SO4', 'Br', 'NO3', 'Al', 'Ba', 'Ca',
      'Fe', 'K', 'Mg', 'Mn', 'Na', 'P', 'Si', 'Sr', 'Ti', 'Li', 'Be', 'Ni',
      'Cu', 'As', 'Cd', 'Tl', 'Pb', 'U', '87Sr/86Sr', 'Benzene', 'Toluene',
      'Ethylbenzene', 'm,p-Xylene', 'o-Xylene', 'Xylene(total)', 'Methane',
      'Ethane', 'Propane', 'δ13C1(‰)', 'δDC1(‰)', 'δ13C2(‰)', 'Radon',
      'Ra-226', 'Ra-228', 'Ra 226 + 228', '228Ra/226Ra'],
      dtype='object')
```

```
In [110... ### All samples
df_well_spring = df_full.drop(columns=['Unnamed: 0', 'Site ID'])
df_well_spring.columns
```

```
Out[110... Index(['DO', 'SC', 'Field pH', 'Lab pH', 'Alkalinity', 'Gas Rover Reading',
      'F', 'Cl', 'SO4', 'Br', 'NO3', 'Al', 'Ba', 'Ca', 'Fe', 'K', 'Mg', 'Mn',
      'Na', 'P', 'Si', 'Sr', 'Ti', 'Li', 'Be', 'Ni', 'Cu', 'As', 'Cd', 'Tl',
      'Pb', 'U', '87Sr/86Sr', 'Benzene', 'Toluene', 'Ethylbenzene',
      'm,p-Xylene', 'o-Xylene', 'Xylene(total)', 'Methane', 'Ethane',
      'Propane', 'δ13C1(‰)', 'δDC1(‰)', 'δ13C2(‰)', 'Radon', 'Ra-226',
      'Ra-228', 'Ra 226 + 228', '228Ra/226Ra'],
      dtype='object')
```

```
In [112... epa_well_spring = epa_table(df_well_spring)
df_well_spring_summary = stats_summary(df_well_spring,df_unit,epa_well_spring)
df_well_spring_summary.to_csv(OUTPUT_DIR/'Full_Table_Stats_Summary(Table 18).csv',index=False)
```

```
In [113... col_label = df_well_spring_summary.columns.tolist()
col_label[-1] = '% Exceeding\n MCL/SMCL'

fig, ax = plt.subplots(figsize=(12, 6)) # Adjust figure size as needed
ax.axis('off') # Remove axes

# Create the table
table = ax.table(cellText=df_well_spring_summary.values, # type: ignore
                 colLabels=col_label,
                 loc='center')

# Customize table appearance
table.auto_set_font_size(False)
table.set_fontsize(8)
table.scale(1.6, 1.5) # Adjust scaling for better spacing

# Export the figure
# plt.savefig(OUTPUT_DIR/'Figures/Well_spring_summary_table_figure.png', bbox_inches='tight')
plt.show()
```

Analyte	Unit	n	Min	Max	Mean	Median	5th Percentile	25th Percentile	75th Percentile	95th Percentile	MCL/SMCL limit	% Exceeding MCL/SMCL
DO	mg/L	90	0.278	9.92	3.955	2.975	0.336	0.669	7.167	9.103	-	-
SC	µS/cm	90	230.0	3485.0	782.881	658.0	428.635	556.25	816.75	1580.8	-	-
pH	pH unit	90	6.05	8.99	7.135	7.04	6.564	6.885	7.278	8.046	6.5-8.5	8.9% (8)
Alkalinity	mg CaCO3/L	80	43.0	668.0	272.449	264.5	170.915	224.75	304.25	442.0	-	-
Air [CH4]	ppm, max	83	5.0	132.0	8.241	5.0	5.0	5.0	5.0	15.9	-	-
F	mg/L	90	0.04	3.52	0.281	0.114	0.057	0.088	0.186	1.331	2	1.1% (1)
Cl	mg/L	90	0.684	697.0	61.616	16.4	2.12	6.498	67.65	254.95	250	5.6% (5)
SO4	mg/L	90	0.06	225.0	40.313	32.6	9.493	23.525	44.15	95.975	250	0.0% (0)
Br	mg/L	90	0.014	5.813	0.125	0.033	0.021	0.027	0.054	0.164	2	1.1% (1)
NO3-NO3	mg/L	90	0.038	27.8	3.653	1.515	0.108	0.3	6.095	12.27	44	0.0% (0)
Al	mg/L	90	0.01	0.02	0.011	0.01	0.01	0.01	0.01	0.02	0.05-0.2	0.0% (0)
Ba	mg/L	90	0.04	2.05	0.17	0.13	0.05	0.082	0.16	0.31	2	1.1% (1)
Ca	mg/L	90	0.67	295.0	84.24	84.3	6.81	56.65	105.5	159.45	-	-
Fe	mg/L	90	0.005	2.12	0.058	0.005	0.005	0.005	0.005	0.196	0.3	4.4% (4)
K	mg/L	90	0.03	7.01	1.989	1.855	0.833	1.368	2.56	3.536	-	-
Mg	mg/L	90	0.31	53.7	16.02	14.85	2.228	10.325	20.375	31.265	-	-
Mn	mg/L	90	0.005	0.43	0.036	0.005	0.005	0.005	0.03	0.195	0.05	16.7% (15)
Na	mg/L	90	2.04	622.0	54.885	15.4	3.436	6.695	57.1	241.35	-	-
P	mg/L	90	0.01	0.29	0.028	0.02	0.01	0.01	0.02	0.07	-	-
Si	mg/L	90	1.05	9.62	5.571	5.32	3.474	4.772	6.212	8.402	-	-
Sr	mg/L	90	0.1	2.14	0.609	0.46	0.14	0.32	0.678	1.648	4	0.0% (0)
Li	µg/L	90	0.27	61.9	11.152	9.86	3.396	6.475	13.175	20.195	10	48.9% (44)
Be	µg/L	90	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	4	0.0% (0)
Ni	µg/L	90	0.05	2.19	0.378	0.275	0.06	0.162	0.46	1.126	100	0.0% (0)
Cu	µg/L	90	0.05	23.2	3.5	2.41	0.059	0.7	4.828	9.772	1000	0.0% (0)
As	µg/L	90	0.05	3.58	0.302	0.11	0.05	0.05	0.18	1.537	10	0.0% (0)
Cd	µg/L	90	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	5	0.0% (0)
Tl	µg/L	90	0.05	1.54	0.091	0.05	0.05	0.05	0.05	0.21	2	0.0% (0)
Pb	µg/L	90	0.05	1.59	0.271	0.185	0.05	0.07	0.358	0.694	15	0.0% (0)
U	µg/L	90	0.05	0.97	0.311	0.26	0.05	0.07	0.478	0.735	30	0.0% (0)
87Sr/86Sr	-	80	0.7111	0.7131	0.7118	0.7117	0.7113	0.7115	0.7121	0.7128	-	-
Benzene	µg/L	84	0.04	0.77	0.113	0.04	0.04	0.04	0.045	0.549	5	0.0% (0)
Toluene	µg/L	84	0.04	3.29	0.432	0.04	0.04	0.04	0.122	3.103	1000	0.0% (0)
Ethylbenzene	µg/L	84	0.04	9.49	1.263	0.04	0.04	0.04	0.11	9.21	700	0.0% (0)
m,p-Xylene	µg/L	84	0.08	0.954	0.105	0.08	0.08	0.08	0.08	0.15	10000	0.0% (0)
o-Xylene	µg/L	84	0.06	0.464	0.074	0.06	0.06	0.06	0.06	0.117	10000	0.0% (0)
Xylene(total)	µg/L	84	0.14	1.418	0.179	0.14	0.14	0.14	0.14	0.267	10000	0.0% (0)
Methane	mg/L	89	0.0	57.0	1.195	0.001	0.0	0.0	0.009	5.48	10	3.4% (3)
Ethane	mg/L	89	0.0	0.58	0.016	0.0	0.0	0.0	0.0	0.032	-	-
Propane	mg/L	89	0.0	0.005	0.0	0.0	0.0	0.0	0.0	0.0	-	-
δ13C1(‰)	-	8	-77.16	-51.5	-66.498	-65.535	-76.145	-71.2	-64.108	-55.7	-	-
δD C1(‰)	-	8	-219.3	-153.9	-191.913	-196.55	-213.945	-202.725	-184.675	-161.18	-	-
δ13C2(‰)	-	1	-35.2	-35.2	-35.2	-35.2	-35.2	-35.2	-35.2	-35.2	-	-
Radon	pCi/L	86	3.74	2760.0	324.719	220.0	20.525	68.375	408.25	979.25	300	38.4% (33)
Ra-226	pCi/L	90	0.028	1.046	0.142	0.096	0.035	0.058	0.162	0.38	27	0.0% (0)
Ra-228	pCi/L	90	0.016	0.864	0.176	0.138	0.036	0.089	0.216	0.472	2.7	0.0% (0)
Ra 226 + 228	-	90	0.055	1.366	0.318	0.24	0.083	0.143	0.387	0.876	5	0.0% (0)
228Ra/226Ra	-	90	0.235	3.23	1.419	1.37	0.5	1.046	1.724	2.53	-	-

Table 1: HS samples

```
In [26]: # Table includes df_full and source/sample group info from df_final
df_extract = pd.merge(df_full,df_final[['Site ID','Source','Sample Group']],on='Site ID')
df_extract.columns
```

```
Out[26]: Index(['Unnamed: 0', 'Site ID', 'DO', 'SC', 'Field pH', 'Lab pH', 'Alkalinity',
               'Gas Rover Reading', 'F', 'Cl', 'SO4', 'Br', 'NO3', 'Al', 'Ba', 'Ca',
               'Fe', 'K', 'Mg', 'Mn', 'Na', 'P', 'Si', 'Sr', 'Ti', 'Li', 'Be', 'Ni',
               'Cu', 'As', 'Cd', 'Tl', 'Pb', 'U', '87Sr/86Sr', 'Benzene', 'Toluene',
               'Ethylbenzene', 'm,p-Xylene', 'o-Xylene', 'Xylene(total)', 'Methane',
               'Ethane', 'Propane', 'δ13C1(‰)', 'δDC1(‰)', 'δ13C2(‰)', 'Radon',
               'Ra-226', 'Ra-228', 'Ra 226 + 228', '228Ra/226Ra', 'Source',
               'Sample Group'],
              dtype='object')
```

```
In [27]: # Generate table 1 & export it before doing stats analysis
df_hs = df_extract[df_extract['Sample Group'].str.contains('HS')]

# # Create groups without spring samples
df_hs_well = df_hs[df_hs['Source'].str.contains('well')]

df_hs = df_hs.drop(columns=['Sample Group','Source','Unnamed: 0', 'Site ID'])
df_hs_well = df_hs_well.drop(columns=['Sample Group','Source','Unnamed: 0', 'Site ID'])

# Generate epa table
epa_hs = epa_table(df_hs)
# Generate stats summary and save it to a csv
df_hs_summary = stats_summary(df_hs,df_unit,epa_hs)
df_hs_summary.to_csv(OUTPUT_DIR/'HS_summary.csv')
```

Table 2: I samples

```
In [28]: df_I = df_extract[df_extract['Sample Group'].str.contains('I')]

# Get samples with only well samples
df_I_well = df_I[df_I['Source'].str.contains('well')]
```

```
df_I = df_I.drop(columns=['Sample Group', 'Source', 'Unnamed: 0', 'Site ID'])
df_I_well = df_I_well.drop(columns=['Sample Group', 'Source', 'Unnamed: 0', 'Site ID'])
# Generate epa table
epa_I = epa_table(df_I)
# Generate stats summary and save it to a csv
df_I_summary = stats_summary(df_I, df_unit, epa_I)
df_I_summary.to_csv(OUTPUT_DIR/'I_summary.csv')
```

Table 3: S samples

```
In [29]: df_s = df_extract[(df_extract['Sample Group']=='S') | (df_extract['Sample Group'].str.contains('HS/S'))]

# Get a table with only well samples
df_s_well = df_s[df_s['Source'].str.contains('well')]
df_s = df_s.drop(columns=['Source', 'Unnamed: 0', 'Sample Group', 'Site ID'])
df_s_well = df_s_well.drop(columns=['Source', 'Unnamed: 0', 'Sample Group', 'Site ID'])
# Generate epa table
epa_s = epa_table(df_s)
# Generate stats summary and save it to a csv
df_s_summary = stats_summary(df_s, df_unit, epa_s)
df_s_summary.to_csv(OUTPUT_DIR/'S_summary.csv')
```

Table 4: PureC Samples

```
In [30]: df_pure_c = df_extract[df_extract['Sample Group'].str.contains('PureC')]
df_well_pC = df_pure_c[df_pure_c['Source'].str.contains('well')]

df_pure_c = df_pure_c.drop(columns=['Sample Group', 'Source', 'Unnamed: 0', 'Site ID'])
df_well_pC = df_well_pC.drop(columns=['Sample Group', 'Source', 'Unnamed: 0', 'Site ID'])

# Generate epa table
epa_pC = epa_table(df_pure_c)
# Generate stats summary and save it to a csv
df_pC_summary = stats_summary(df_pure_c, df_unit, epa_pC)
df_pC_summary.to_csv(OUTPUT_DIR/'PureC_summary.csv')
```

OLS + Spearman + Kendall's Tau (non-parametric correlation) + ATS slope

```
In [61]: # Create a function that calculate the kendall's Tau, p-value nad ATS slope between two dataset

from scipy.stats import kendalltau
from scipy.stats import theilslopes
from scipy.stats import spearmanr
from scipy.stats import linregress

def Correlation_test(df, feature): # df is the analytes' columns and feature is geospatial features (e.g., distance to

    # Store the results
    result = {}
    significance_threshold = 0.05

    for column in df.columns:

        # Drop na from the columns
        temp_df = pd.concat([df[column], feature], axis=1).dropna()
        col1 = temp_df[column]
        col2 = temp_df[feature.name]

        # Calculate linear regression
        if 'DIST' in feature.name:

            # Log transformation to the concentration data
            # col1 = np.log(col1+1)

            # turn unit of feature from m to km

            col2 = col2/1000

            # Calculate linear regression
            ols_slope, ols_intercept, ols_r_value, ols_p_value, ols_std_err = linregress(col2, col1)
        else:
            ols_slope = np.nan
```

```

    ols_p_value = np.nan
    # Calculate spearman correlation
    spearman_corr, spearman_p = spearmanr(col1, col2)
    # Calculate Kendall's Tau
    tau, p_value = kendalltau(col1, col2)
    # Calculat ATS slopes
    slope, intercept, lo_slope, hi_slope = theilslopes(col1, col2, 0.95)
    # flag the significant correlations
    result[column] = {
        'OLS_slope': f'{ols_slope:.3f}' + ('*' if ols_p_value < significance_threshold else ''), # + f'({ols_p_value:.3f})',
        'Spearman': f'{spearman_corr:.3f}' + ('*' if spearman_p < significance_threshold else ''), # + f'({spearman_p:.3f})',
        'Kendall\'s Tau': f'{tau:.3f}' + ('*' if p_value < significance_threshold else ''), # + f'({p_value:.3f})',
        'ATS slope': f'{slope:.3f}' + ('*' if p_value < significance_threshold else ''), # + f'({p_value:.3f})',
    }
    result = pd.DataFrame(result)
    print(result)
    return result

```

```

In [62]: df_corr = pd.merge(df_full, df_final[['Site ID', 'Source',
        'NEAR_UOG_DIST', 'Impoundment_DIST', 'Recent_Spill_DIST']], on='Site ID')

df_corr_well = df_corr[df_corr['Source'].str.contains('well')]

df_corr.columns

```

```

Out[62]: Index(['Unnamed: 0', 'Site ID', 'DO', 'SC', 'Field pH', 'Lab pH', 'Alkalinity',
        'Gas Rover Reading', 'F', 'Cl', 'SO4', 'Br', 'NO3', 'Al', 'Ba', 'Ca',
        'Fe', 'K', 'Mg', 'Mn', 'Na', 'P', 'Si', 'Sr', 'Ti', 'Li', 'Be', 'Ni',
        'Cu', 'As', 'Cd', 'Tl', 'Pb', 'U', '87Sr/86Sr', 'Benzene', 'Toluene',
        'Ethylbenzene', 'm,p-Xylene', 'o-Xylene', 'Xylene(total)', 'Methane',
        'Ethane', 'Propane', 'δ13C1(‰)', 'δDC1(‰)', 'δ13C2(‰)', 'Radon',
        'Ra-226', 'Ra-228', 'Ra 226 + 228', '228Ra/226Ra', 'Source',
        'NEAR_UOG_DIST', 'Impoundment_DIST', 'Recent_Spill_DIST'],
        dtype='object')

```

```

In [63]: # After Cell 38 – check for further inflation
print("df_corr shape:", df_corr.shape)
print("Unique Site IDs in df_corr:", df_corr['Site ID'].nunique(),
        "vs total rows:", len(df_corr))

```

```

df_corr shape: (90, 56)
Unique Site IDs in df_corr: 90 vs total rows: 90

```

```
In [73]: df_analyte = df_corr[['SC', 'Na', 'Cl', 'Ba', 'SO4', 'Sr', 'Br', 'Fe', 'Methane', 'Ra-226', 'Ra-228', 'Radon']]
df_analyte_well = df_corr_well[['SC', 'Na', 'Cl', 'Ba', 'SO4', 'Sr', 'Br', 'Fe', 'Methane', 'Ra-226', 'Ra-228', 'Radon']]

# Change the name of 'Radon' to 'Rn'
df_analyte = df_analyte.rename(columns={'Radon': 'Rn'})
df_analyte_well = df_analyte_well.rename(columns={'Radon': 'Rn'})
```

```
In [74]: df_analyte
```

```
Out[74]:
```

	SC	Na	Cl	Ba	SO4	Sr	Br	Fe	Methane	Ra-226	Ra-228	Rn
0	1011.0	267.00	4.79	0.07	13.30	0.11	0.033981	0.005	0.00120	0.055298	0.016092	40.0
1	608.0	7.99	28.40	0.06	21.80	0.46	0.186893	0.005	0.00020	0.052514	0.045508	2760.0
2	501.0	6.39	12.50	0.16	25.30	0.40	0.022654	0.005	0.00062	0.170056	0.211928	393.0
3	530.0	18.00	18.10	0.12	32.10	0.63	0.121359	0.005	0.00020	0.092020	0.163812	126.0
4	957.0	65.40	128.00	0.24	18.20	2.14	0.067961	0.010	0.00180	0.135765	0.253120	78.1
...	...	...	...	...	...	...	...	...	...	...	...	...
85	563.0	18.90	26.40	0.81	9.34	0.61	0.015261	0.860	1.50000	0.322592	0.167780	213.0
86	551.0	47.30	80.70	0.07	47.80	0.25	0.033735	0.020	0.00310	0.164519	0.173137	472.0
87	1056.0	12.70	122.00	0.04	82.50	0.78	0.055422	0.005	0.04100	0.055547	0.059220	NaN
88	550.0	44.50	4.06	0.14	27.70	0.78	0.033000	0.020	0.20000	0.231318	0.265451	NaN
89	431.0	15.30	6.19	0.10	28.90	0.39	0.029719	0.005	0.00910	0.054557	0.122037	NaN

90 rows × 12 columns

```
In [75]: df_analyte.columns
```

```
Out[75]: Index(['SC', 'Na', 'Cl', 'Ba', 'SO4', 'Sr', 'Br', 'Fe', 'Methane', 'Ra-226',
               'Ra-228', 'Rn'],
              dtype='object')
```

Analyte concentrations vs. Groups of interest - Well + Spring

Table 8

```
In [76]: # Hotspot
corr_hotspot = Correlation_test(df_analyte, df_corr['NEAR_UOG_DIST'])

corr_hotspot.to_csv(OUTPUT_DIR/'Correlation_hotspot(Table 8).csv', header=True, index=True)
```

	SC	Na	Cl	Ba	SO4	Sr	Br	Fe \
OLS_slope	2.930	3.055*	0.119	0.003	-0.588	-0.005	0.012	0.002
Spearman	-0.099	0.026	-0.051	-0.041	-0.206	-0.109	0.086	0.139
Kendall's Tau	-0.071	0.022	-0.038	-0.029	-0.133	-0.069	0.058	0.104
ATS slope	-3.793	0.121	-0.192	0.000	-0.686	-0.006	0.000	0.000

	Methane	Ra-226	Ra-228	Rn
OLS_slope	0.206*	0.001	0.002	-5.476
Spearman	0.310*	0.044	0.006	-0.113
Kendall's Tau	0.214*	0.030	0.007	-0.082
ATS slope	0.000*	0.001	0.000	-6.490

Table 9

```
In [77]: # Impoundment
result_impoundment = Correlation_test(df_analyte, df_corr['Impoundment_DIST'])

result_impoundment.to_csv(OUTPUT_DIR/'Correlation_impoundment(Table 9).csv', header=True, index=True)
```

	SC	Na	Cl	Ba	SO4	Sr	Br	Fe \
OLS_slope	11.504	4.617*	1.860	0.007	0.077	-0.005	0.029*	-0.002
Spearman	-0.063	0.050	-0.085	-0.129	0.028	-0.225*	-0.051	-0.156
Kendall's Tau	-0.034	0.033	-0.050	-0.086	0.030	-0.151*	-0.037	-0.117
ATS slope	-2.310	0.170	-0.236	-0.001	0.160	-0.014*	-0.000	0.000

	Methane	Ra-226	Ra-228	Rn
OLS_slope	0.097	-0.000	-0.000	-8.447
Spearman	0.144	-0.162	-0.192	0.010
Kendall's Tau	0.095	-0.124	-0.133	0.010
ATS slope	0.000	-0.003	-0.004	0.447

Table 10

```
In [78]: # Spill
result_spill = Correlation_test(df_analyte, df_corr['Recent_Spill_DIST'])

result_spill.to_csv(OUTPUT_DIR/'Correlation_spill(Table 10).csv', header=True, index=True)
```

	SC	Na	Cl	Ba	SO4	Sr	Br	\
OLS_slope	-15.816	-5.954*	-4.165	-0.014*	1.072	-0.011	-0.026	
Spearman	0.000	-0.307*	-0.119	-0.084	0.208*	-0.048	-0.101	
Kendall's Tau	-0.002	-0.209*	-0.084	-0.054	0.145*	-0.037	-0.069	
ATS slope	-0.143	-1.199*	-0.521	-0.001	1.106*	-0.005	-0.000	

	Fe	Methane	Ra-226	Ra-228	Rn
OLS_slope	-0.005	-0.286	-0.004	-0.004	3.651
Spearman	0.034	-0.259*	-0.085	-0.110	-0.025
Kendall's Tau	0.024	-0.172*	-0.058	-0.075	-0.019
ATS slope	0.000	-0.000*	-0.002	-0.003	-1.286

Analyte concentrations vs. Groups of interest - Well only

Table 15

```
In [79]: # Hotspot
corr_hotspot_well = Correlation_test(df_analyte_well, df_corr_well['NEAR_UOG_DIST'])

corr_hotspot_well.to_csv(OUTPUT_DIR/'Correlation_hotspot_well_only(Table 15).csv', header=True, index=True)
```

	SC	Na	Cl	Ba	SO4	Sr	Br	Fe \
OLS_slope	3.401	3.078	-0.100	0.004	-0.596	-0.006	0.013	0.002
Spearman	-0.066	0.027	-0.050	0.060	-0.172	-0.102	0.112	0.090
Kendall's Tau	-0.044	0.028	-0.037	0.038	-0.106	-0.062	0.071	0.064
ATS slope	-2.638	0.191	-0.258	0.000	-0.573	-0.005	0.000	0.000

	Methane	Ra-226	Ra-228	Rn
OLS_slope	0.217	0.001	0.002	-10.272
Spearman	0.318*	0.139	0.076	-0.175
Kendall's Tau	0.218*	0.099	0.055	-0.119
ATS slope	0.000*	0.002	0.002	-9.278

Table 16

```
In [80]: # Impoundment
corr_impoundment_well = Correlation_test(df_analyte_well, df_corr_well['Impoundment_DIST'])

corr_impoundment_well.to_csv(OUTPUT_DIR/'Correlation_impoundment_well_only(Table 16).csv')
```

	SC	Na	Cl	Ba	SO4	Sr	Br	Fe \
OLS_slope	13.670	4.994*	2.039	0.008	0.122	-0.006	0.031*	-0.002
Spearman	-0.005	0.087	-0.074	-0.102	0.055	-0.261*	-0.018	-0.205
Kendall's Tau	0.001	0.057	-0.039	-0.067	0.046	-0.172*	-0.016	-0.156
ATS slope	0.000	0.373	-0.213	-0.001	0.269	-0.019*	-0.000	0.000

	Methane	Ra-226	Ra-228	Rn
OLS_slope	0.103	-0.001	-0.000	-17.963
Spearman	0.103	-0.170	-0.197	-0.110
Kendall's Tau	0.059	-0.129	-0.137	-0.072
ATS slope	0.000	-0.004	-0.005	-5.574

Table 17

```
In [81]: # Spill
corr_spill_well = Correlation_test(df_analyte_well, df_corr_well['Recent_Spill_DIST'])

corr_spill_well.to_csv(OUTPUT_DIR/'Correlation_spill_well_only(Table 17).csv')
```

	SC	Na	Cl	Ba	S04	Sr	Br \
OLS_slope	-19.252	-6.885*	-4.563	-0.017*	1.231	-0.012	-0.031
Spearman	-0.022	-0.322*	-0.066	-0.138	0.212	-0.054	-0.095
Kendall's Tau	-0.016	-0.221*	-0.047	-0.094	0.146	-0.041	-0.067
ATS slope	-0.987	-1.640*	-0.324	-0.002	1.247	-0.006	-0.000

	Fe	Methane	Ra-226	Ra-228	Rn
OLS_slope	-0.006	-0.335	-0.004	-0.005	8.716
Spearman	0.040	-0.314*	-0.123	-0.145	0.044
Kendall's Tau	0.028	-0.215*	-0.082	-0.100	0.032
ATS slope	0.000	-0.000*	-0.002	-0.003	2.785

One-sided test with Benjamin - Hochberg (BH) procedure

Define Functions

```
In [43]: ### Analyte for comparison
analyte_list = ['D0', 'SC', 'F', 'Cl', 'S04', 'Br', 'N03', 'Al', 'Ba', 'Ca', 'Fe', 'K',
                'Mg', 'Mn', 'Na', 'P', 'Si', 'Sr', 'Li', 'Be', 'Ni', 'Cu', 'As', 'Cd',
                'Tl', 'Pb', 'U', 'Benzene', 'Toluene', 'Ethylbenzene', 'm,p-Xylene',
                'o-Xylene', 'Xylene(total)', 'Methane', 'Ethane', 'Propane', 'Radon', 'Ra-226',
                'Ra-228', 'Ra 226 + 228', '228Ra/226Ra']
```

```
In [44]: # Define one-sided test with BH in a function
from scipy.stats import mannwhitneyu
from scipy.stats import brunnermunzel
from statsmodels.stats.multitest import multipletests

def one_sided_bh_wmw(df_1, df_2):

    # Store the p-value for bh correction
    p_values_greater = []
    p_values_less = []
    median_diff = []

    significance_threshold = 0.05

    for feature in df_1.columns:
```

```

# Create copies of the columns for comparison
col1 = df_1[feature].dropna()
col2 = df_2[feature].dropna()

# Test if Group A has higher values
stat_greater, p_value_greater = mannwhitneyu(col1, col2, alternative='greater')
p_values_greater.append(p_value_greater)

# Test if Group A has lower values
stat_less, p_value_less = mannwhitneyu(col1, col2, alternative='less')
p_values_less.append(p_value_less)

# Calculate the median difference
median_diff.append(col1.median() - col2.median())

### Applying bh ###
p_values_greater = np.array(p_values_greater)
p_values_less = np.array(p_values_less)
median_diff = np.array(median_diff)

# Combine p-values
all_pvalues = np.concatenate([p_values_greater, p_values_less])

# Apply BH correction to all 40 tests
fdr_level = 0.1
rejected, p_adjusted, _, _ = multipletests(all_pvalues, alpha=fdr_level, method='fdr_bh')

results_df = pd.DataFrame({
    'analyte': np.tile(df_1.columns, 2),
    'test': ['left > right']*df_1.shape[1] + ['left < right']*df_1.shape[1],
    'original_pvalue': all_pvalues,
    'adjusted_pvalue': p_adjusted,
    'significant': rejected
})

# Display the results
significant_results = results_df[results_df['significant']]
one_sided_results = results_df[results_df['original_pvalue'] < 0.05]

```

```

print("One-Sided Test Results for Significant Features:")
print(significant_results)
print("\n")
print("Without fdr:", '\n')
print(one_sided_results)

return significant_results, one_sided_results

```

```

In [45]: def one_sided_bh_bm(df_1, df_2):

    # Store the p-value for bh correction
    p_values_greater = []
    p_values_less = []
    median_diff = []

    significance_threshold = 0.05

    for feature in df_1.columns:
        # Create copies of the columns for comparison
        col1 = df_1[feature].dropna()
        col2 = df_2[feature].dropna()

        # Test if Group A has higher values
        stat_greater, p_value_greater = brunnermunzel(col1, col2, alternative='greater')
        p_values_greater.append(p_value_greater)

        # Test if Group A has lower values
        stat_less, p_value_less = brunnermunzel(col1, col2, alternative='less')
        p_values_less.append(p_value_less)

        # Calculate the median difference
        median_diff.append(col1.median() - col2.median())

    ### Applying bh ###
    p_values_greater = np.array(p_values_greater)
    p_values_less = np.array(p_values_less)
    median_diff = np.array(median_diff)

    # Combine p-values
    all_pvalues = np.concatenate([p_values_greater, p_values_less])

```

```

all_pvalues[np.isnan(all_pvalues)] = 1

# Apply BH correction to all 40 tests
fdr_level = 0.1
rejected, p_adjusted, _, _ = multipletests(all_pvalues, alpha=fdr_level, method='fdr_bh')

results_df = pd.DataFrame({
    'analyte': np.tile(df_1.columns, 2),
    'test': ['left > right']*df_1.shape[1] + ['left < right']*df_1.shape[1],
    'original_pvalue': all_pvalues,
    'adjusted_pvalue': p_adjusted,
    'significant': rejected
})

# Display the results
significant_results = results_df[results_df['significant']]
one_sided_results = results_df[results_df['original_pvalue'] < 0.05]

print("One-Sided Test Results for Significant Features:")
print(significant_results)
print("\n")
print("Without fdr:", '\n')
print(one_sided_results)

return significant_results, one_sided_results

```

```

In [46]: ### Before reperforming the test, we setup the dataframes for comparison
df_HS = df_hs.loc[:, analyte_list]
df_I = df_I.loc[:, analyte_list]
df_S = df_s.loc[:, analyte_list]
df_pure_c = df_pure_c.loc[:, analyte_list]

```

One-sided tests: Well & Spring

Table 4 & Table 4A

```

In [47]: # df_HS vs df_pure_c
sig, one_sided_bh = one_sided_bh_wmw(df_HS, df_pure_c)

```

```
one_sided_bh.to_csv(OUTPUT_DIR/'one_sided_wmw_HS_pure_c.csv')
```

One-Sided Test Results for Significant Features:

Empty DataFrame

Columns: [analyte, test, original_pvalue, adjusted_pvalue, significant]

Index: []

Without fdr:

	analyte	test	original_pvalue	adjusted_pvalue	significant
6	NO3	left > right	0.020705	0.242544	False
11	K	left > right	0.012023	0.242544	False
26	U	left > right	0.043942	0.399135	False
36	Radon	left > right	0.017729	0.242544	False
37	Ra-226	left > right	0.018863	0.242544	False
38	Ra-228	left > right	0.010402	0.242544	False
39	Ra 226 + 228	left > right	0.011538	0.242544	False
56	P	left < right	0.024900	0.255229	False
70	Ethylbenzene	left < right	0.020173	0.242544	False

```
In [48]: sig, one_sided_bh = one_sided_bh_bm(df_HS, df_pure_c)
```

```
one_sided_bh.to_csv(OUTPUT_DIR/'one_sided_bm_HS_pure_c.csv')
```

One-Sided Test Results for Significant Features:

	analyte	test	original_pvalue	adjusted_pvalue	significant
6	NO3	left > right	0.004696	0.083416	True
11	K	left > right	0.005086	0.083416	True
37	Ra-226	left > right	0.002164	0.059137	True
38	Ra-228	left > right	0.001060	0.043475	True
39	Ra 226 + 228	left > right	0.000831	0.043475	True

Without fdr:

	analyte	test	original_pvalue	adjusted_pvalue	significant
3	Cl	left > right	0.047189	0.418201	False
6	NO3	left > right	0.004696	0.083416	True
11	K	left > right	0.005086	0.083416	True
26	U	left > right	0.032108	0.329109	False
36	Radon	left > right	0.008344	0.114039	False
37	Ra-226	left > right	0.002164	0.059137	True
38	Ra-228	left > right	0.001060	0.043475	True
39	Ra 226 + 228	left > right	0.000831	0.043475	True
56	P	left < right	0.025539	0.299177	False

```

c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)

```

```
In [49]: # df_I vs df_pure_c
sig, one_sided_bh = one_sided_bh_wmw(df_I, df_pure_c)

one_sided_bh.to_csv(OUTPUT_DIR/'one_sided_wmw_I_pure_c.csv')
```

One-Sided Test Results for Significant Features:

Empty DataFrame

Columns: [analyte, test, original_pvalue, adjusted_pvalue, significant]

Index: []

Without fdr:

	analyte	test	original_pvalue	adjusted_pvalue	significant
3	Cl	left > right	0.035182	0.480827	False
25	Pb	left > right	0.018022	0.295555	False
37	Ra-226	left > right	0.008813	0.240902	False
38	Ra-228	left > right	0.002792	0.155654	False
39	Ra 226 + 228	left > right	0.003796	0.155654	False
56	P	left < right	0.015993	0.295555	False

```
In [50]: sig, one_sided_bh = one_sided_bh_bm(df_I, df_pure_c)

one_sided_bh.to_csv(OUTPUT_DIR/'one_sided_bm_I_pure_c.csv')
```

One-Sided Test Results for Significant Features:

	analyte	test	original_pvalue	adjusted_pvalue	significant
37	Ra-226	left > right	0.000869	0.023762	True
38	Ra-228	left > right	0.000078	0.005681	True
39	Ra 226 + 228	left > right	0.000139	0.005681	True

Without fdr:

	analyte	test	original_pvalue	adjusted_pvalue	significant
25	Pb	left > right	0.019840	0.271140	False
27	Benzene	left > right	0.018612	0.271140	False
37	Ra-226	left > right	0.000869	0.023762	True
38	Ra-228	left > right	0.000078	0.005681	True
39	Ra 226 + 228	left > right	0.000139	0.005681	True
56	P	left < right	0.014424	0.271140	False

```

c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)

```

```

In [51]: # df_S vs df_pure_c
sig, one_sided_bh = one_sided_bh_wmw(df_S, df_pure_c)

one_sided_bh.to_csv(OUTPUT_DIR/'one_sided_wmw_S_pure_c.csv')

```

One-Sided Test Results for Significant Features:

Empty DataFrame

Columns: [analyte, test, original_pvalue, adjusted_pvalue, significant]

Index: []

Without fdr:

	analyte	test	original_pvalue	adjusted_pvalue	significant
11	K	left > right	0.029687	0.608583	False
21	Cu	left > right	0.023859	0.608583	False
36	Radon	left > right	0.048940	0.644163	False
38	Ra-228	left > right	0.007971	0.608583	False
39	Ra 226 + 228	left > right	0.017585	0.608583	False

```
In [52]: sig, one_sided_bh = one_sided_bh_bm(df_S, df_pure_c)

one_sided_bh.to_csv(OUTPUT_DIR/'one_sided_bh_S_pure_c.csv')
```

One-Sided Test Results for Significant Features:

Empty DataFrame

Columns: [analyte, test, original_pvalue, adjusted_pvalue, significant]

Index: []

Without fdr:

	analyte	test	original_pvalue	adjusted_pvalue	significant
11	K	left > right	0.018059	0.370212	False
21	Cu	left > right	0.013981	0.370212	False
36	Radon	left > right	0.035047	0.502148	False
37	Ra-226	left > right	0.036743	0.502148	False
38	Ra-228	left > right	0.001541	0.126322	False
39	Ra 226 + 228	left > right	0.006762	0.277246	False

```

c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)

```

One-sided tests: Well only

```

In [53]: df_hs_well = df_hs_well.loc[:, analyte_list]
df_i_well = df_i_well.loc[:, analyte_list]
df_s_well = df_s_well.loc[:, analyte_list]
df_pC_well = df_well_pC.loc[:, analyte_list]

```

Table 5 & 5A

```
In [54]: # df_hs_well vs df_well_pC
sig, one_sided_bh = one_sided_bh_wmw(df_hs_well, df_pC_well)

one_sided_bh.to_csv(OUTPUT_DIR/'one_sided_wmw_hs_well_only.csv')
```

One-Sided Test Results for Significant Features:

Empty DataFrame

Columns: [analyte, test, original_pvalue, adjusted_pvalue, significant]

Index: []

Without fdr:

	analyte	test	original_pvalue	adjusted_pvalue	significant
3	Cl	left > right	0.034476	0.353383	False
11	K	left > right	0.012798	0.258342	False
12	Mg	left > right	0.040851	0.372195	False
36	Radon	left > right	0.029652	0.347357	False
37	Ra-226	left > right	0.018672	0.258342	False
38	Ra-228	left > right	0.012421	0.258342	False
39	Ra 226 + 228	left > right	0.012421	0.258342	False
56	P	left < right	0.018903	0.258342	False
70	Ethylbenzene	left < right	0.016161	0.258342	False

```
In [55]: sig, one_sided_bh = one_sided_bh_bm(df_hs_well, df_pC_well)

one_sided_bh.to_csv(OUTPUT_DIR/'one_sided_bm_hs_well_only.csv')
```

One-Sided Test Results for Significant Features:

	analyte	test	original_pvalue	adjusted_pvalue	significant
37	Ra-226	left > right	0.002429	0.066405	True
38	Ra-228	left > right	0.001357	0.055627	True
39	Ra 226 + 228	left > right	0.001025	0.055627	True

Without fdr:

	analyte	test	original_pvalue	adjusted_pvalue	significant
3	Cl	left > right	0.034198	0.377046	False
11	K	left > right	0.005880	0.120530	False
12	Mg	left > right	0.036785	0.377046	False
36	Radon	left > right	0.012263	0.201119	False
37	Ra-226	left > right	0.002429	0.066405	True
38	Ra-228	left > right	0.001357	0.055627	True
39	Ra 226 + 228	left > right	0.001025	0.055627	True
56	P	left < right	0.020282	0.277187	False
70	Ethylbenzene	left < right	0.043820	0.399251	False

```

c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kwds)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kwds)

```

```

In [56]: # df_i_well vs df_well_pC
sig, one_sided_bh = one_sided_bh_wmw(df_i_well, df_pC_well)

```

```
one_sided_bh.to_csv(OUTPUT_DIR/'one_sided_wmw_i_well_only.csv')
```

One-Sided Test Results for Significant Features:

	analyte	test	original_pvalue	adjusted_pvalue	significant
38	Ra-228	left > right	0.001663	0.084998	True
39	Ra 226 + 228	left > right	0.002073	0.084998	True

Without fdr:

	analyte	test	original_pvalue	adjusted_pvalue	significant
3	Cl	left > right	0.040634	0.555328	False
25	Pb	left > right	0.027898	0.457527	False
37	Ra-226	left > right	0.005612	0.153406	False
38	Ra-228	left > right	0.001663	0.084998	True
39	Ra 226 + 228	left > right	0.002073	0.084998	True
56	P	left < right	0.020618	0.422661	False

```
In [57]: sig, one_sided_bh = one_sided_bh_bm(df_i_well, df_pC_well)
```

```
one_sided_bh.to_csv(OUTPUT_DIR/'one_sided_bm_i_well_only.csv')
```

One-Sided Test Results for Significant Features:

	analyte	test	original_pvalue	adjusted_pvalue	significant
37	Ra-226	left > right	0.000269	0.007341	True
38	Ra-228	left > right	0.000014	0.001110	True
39	Ra 226 + 228	left > right	0.000027	0.001110	True

Without fdr:

	analyte	test	original_pvalue	adjusted_pvalue	significant
25	Pb	left > right	0.029627	0.485886	False
37	Ra-226	left > right	0.000269	0.007341	True
38	Ra-228	left > right	0.000014	0.001110	True
39	Ra 226 + 228	left > right	0.000027	0.001110	True
56	P	left < right	0.020905	0.428549	False

```

c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)

```

```

In [58]: # df_s_well vs df_pC_well
sig, one_sided_bh = one_sided_bh_wmw(df_s_well, df_pC_well)

one_sided_bh.to_csv(OUTPUT_DIR/'one_sided_wmw_s_well_only.csv')

```

One-Sided Test Results for Significant Features:

Empty DataFrame

Columns: [analyte, test, original_pvalue, adjusted_pvalue, significant]

Index: []

Without fdr:

	analyte	test	original_pvalue	adjusted_pvalue	significant
11	K	left > right	0.048400	0.897903	False
21	Cu	left > right	0.021616	0.760905	False
38	Ra-228	left > right	0.018473	0.760905	False
39	Ra 226 + 228	left > right	0.027838	0.760905	False

```
In [59]: sig, one_sided_bh = one_sided_bh_bm(df_s_well, df_pC_well)

one_sided_bh.to_csv(OUTPUT_DIR/'one_sided_bm_s_well_only.csv')
```

```
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8819:
RuntimeWarning: invalid value encountered in scalar divide
    wbfm /= (nx + ny) * np.sqrt(nx * Sx + ny * Sy)
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_stats_py.py:8825:
RuntimeWarning: invalid value encountered in scalar divide
    df = df_numer / df_denom
c:\Apps-SU\HEI_CODE_BASE\HEI_DOC_CODE\HEI_Script_Data_Results\.venv\Lib\site-packages\scipy\stats\_axis_nan_policy.p
y:586: RuntimeWarning: p-value cannot be estimated with `distribution='t'` because degrees of freedom parameter is und
efined (0/0). Try using `distribution='normal'`
    res = hypotest_fun_out(*samples, **kws)
```

One-Sided Test Results for Significant Features:

Empty DataFrame

Columns: [analyte, test, original_pvalue, adjusted_pvalue, significant]

Index: []

Without fdr:

	analyte	test	original_pvalue	adjusted_pvalue	significant
11	K	left > right	0.029389	0.602479	False
21	Cu	left > right	0.013803	0.377289	False
38	Ra-228	left > right	0.006696	0.377289	False
39	Ra 226 + 228	left > right	0.012159	0.377289	False

In []: