IR Spectroscopy Analyzer

Group 12

Asia 黃夏彤 Charoline 黃菊花 Teresa 陳翠珍 Thalia 郭伶美

Basic concepts

What is IR spectroscopy?

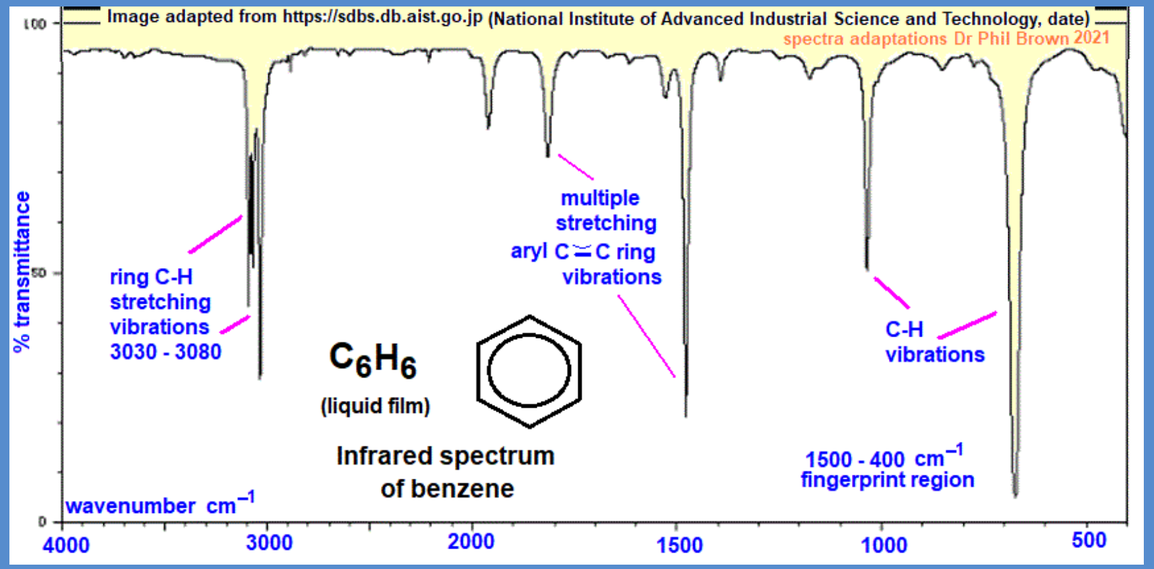
A method used in analytical chemistry to predict a chemical structure using infrared electromagnetic waves

Usually, IR is used to confirm the existence of a functional group for chemical synthesis!

How it works?

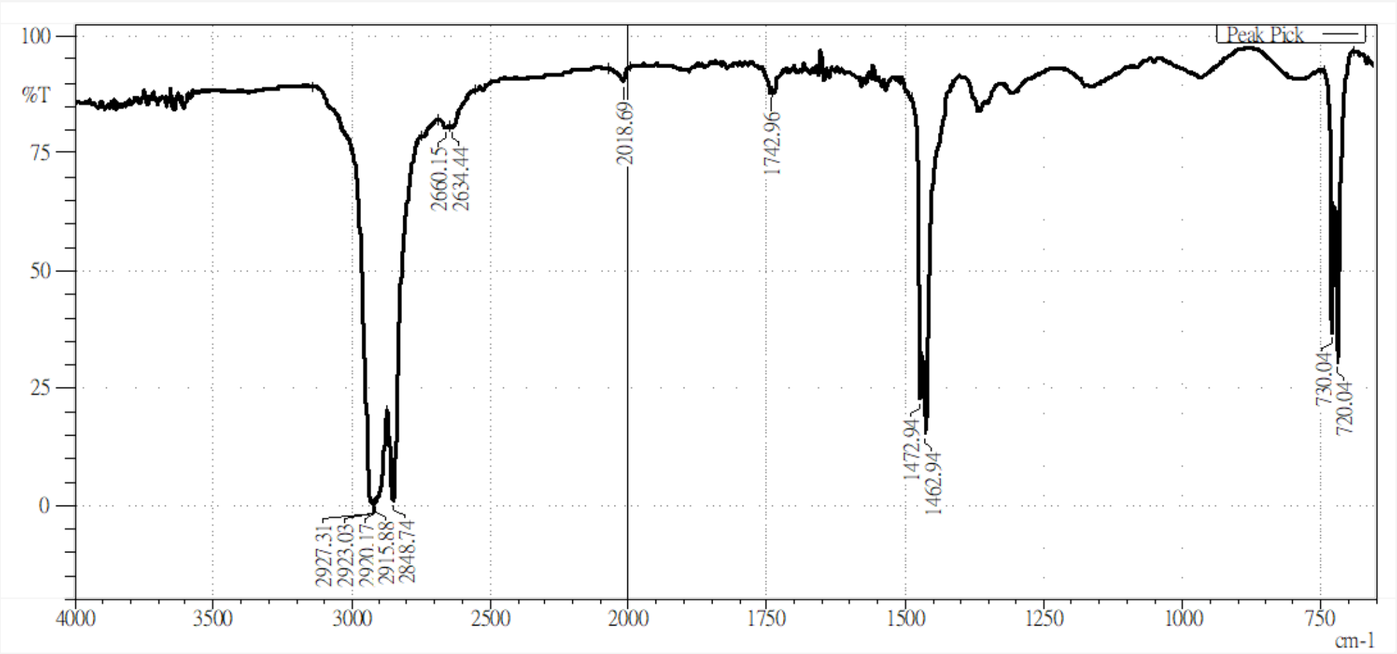
Different functional groups and bonds affect the waves' absorbance at various wavelengths. This data is then plotted and used to analyze the compound's structure!

Functional group	Mode of vibration		Wavenumber (cm^{-1})	Intensity
C–H	Alkanes	(stretching)	3000-2850	Strong
	$-CH_3$	(bending)	1450 and 1375	Medium
	$-CH_2-$	(bending)	1465	Medium
	Alkenes	(stretching)	3100-3000	Medium
		(out of plane bending)	1000-650	Strong
	Aromatics	(stretching)	3150-3050	Strong
		(out of plane bending)	900-690	Strong
	Alkyne	(stretching)	ca. 3300	Strong
	Aldehyde		2900-2800	Weak
			2800-2700	Weak
C=C	Alkane	not interpretative useful		
	Alkene		1680-1600	Medium to Weak
	Aromatic		1600 and 1475	Medium to Weak
C≡C	Alkyne		2250-2100	Medium to Weak



Why?

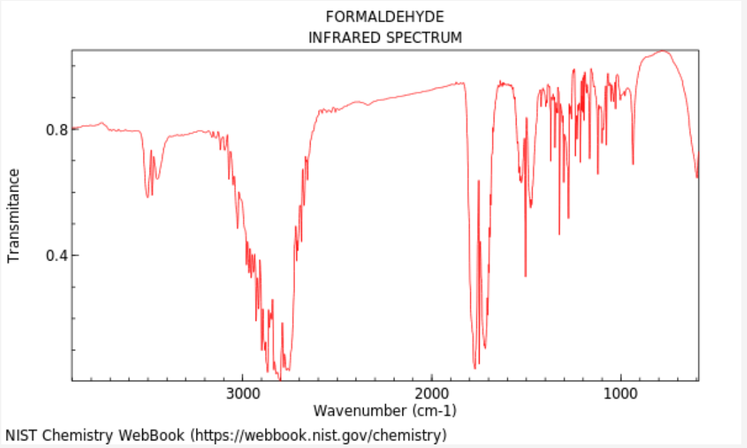
- Manually pick out which peak to use -> bias
- Sometimes peaks are too close -> hard to notice
- Checking the table now and then is time-consuming

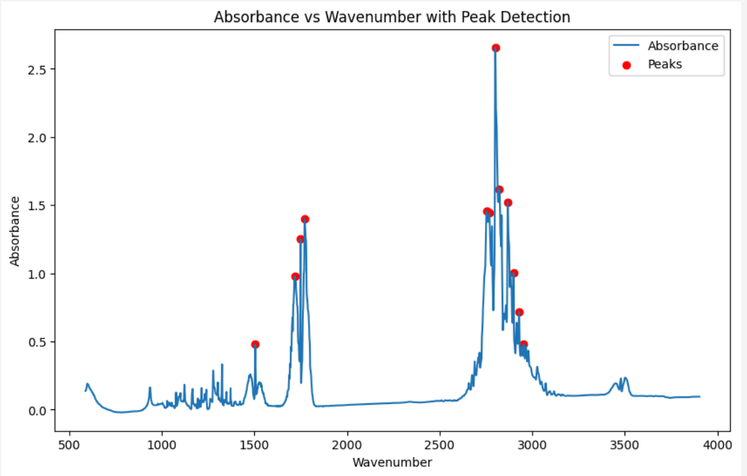


Preprocessing Data

The original data from IR spectroscopy is typically presented in terms of transmittance, which can contain significant biases. Therefore, the raw data is processed using the equation

$$A = -log_{10}(T)$$





Purposes:

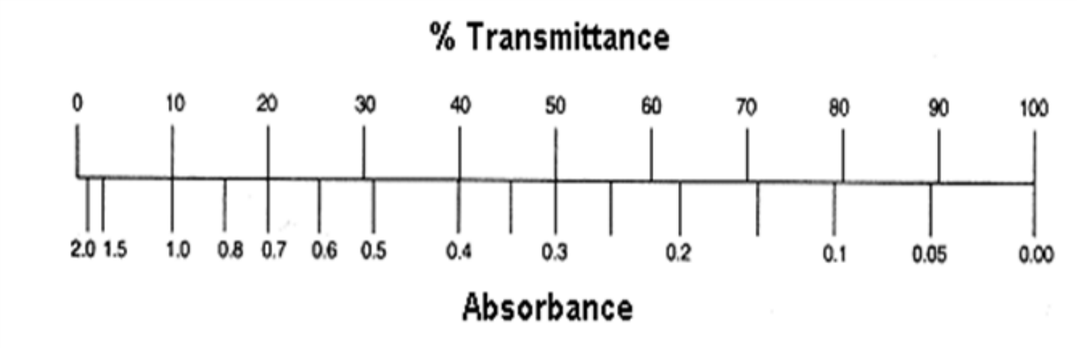
1. **Transforms the relationship**: It converts transmittance (which can decrease in a valley-like shape) into absorbance
2. **Enhances clarity**: The logarithmic function magnifies differences in low transmittance values, making small features or variations in the data more visible and distinct

Reference : Ramadhani, F., et al (2019). Optical properties of dye DN-F01 as sensitizer. Journal of Physics Theories and Applications, 3(1), 36.

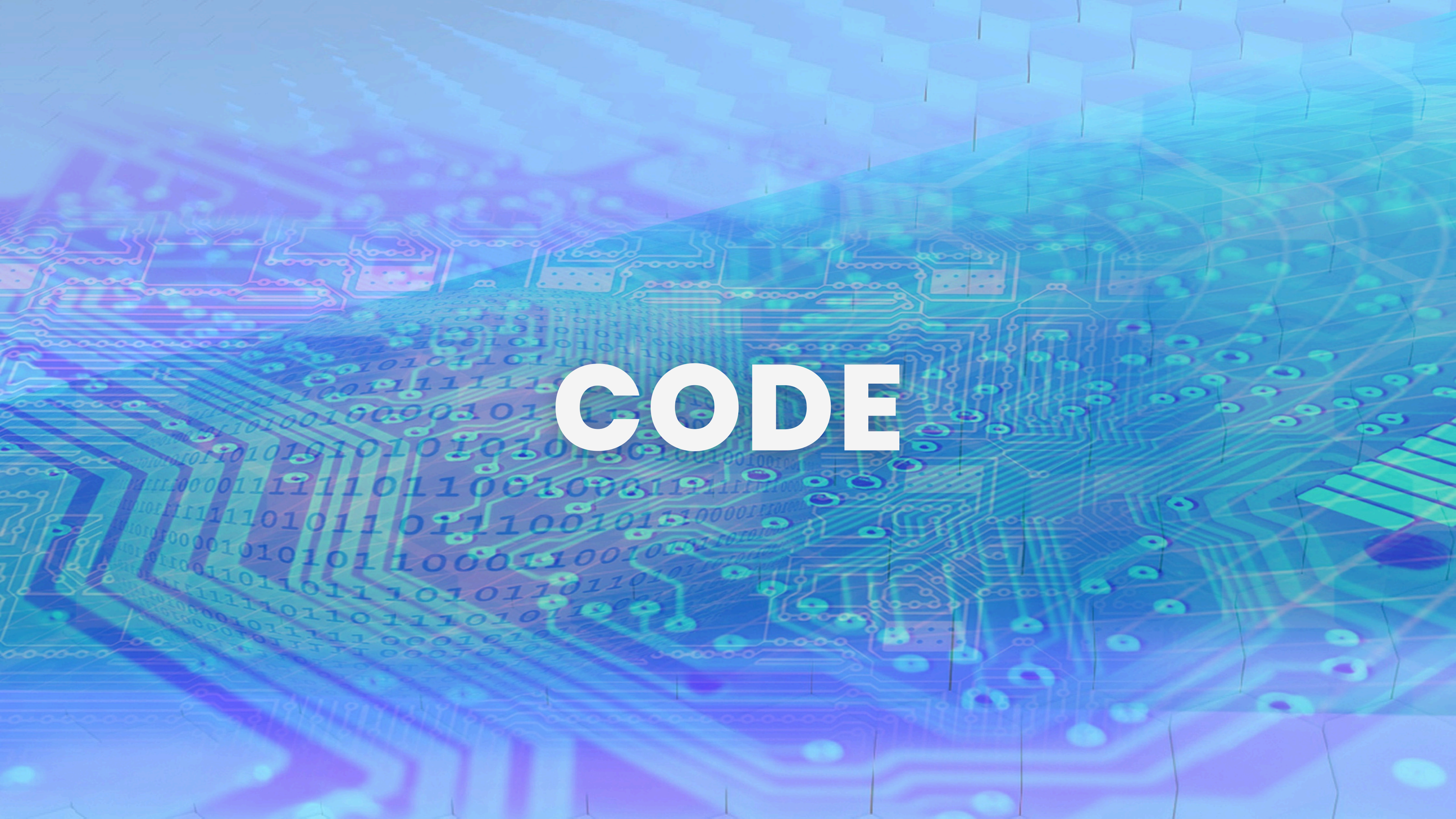
Preprocessing Data

$$A = -log_{10}(T)$$

Relationship of Absorbance and Transmittance



Reference : Ramadhani, F., et al (2019). Optical properties of dye DN-F01 as sensitizer. Journal of Physics Theories and Applications, 3(1), 36.


CODE

Code Overview

File Conversion

JDX FILE
X1 Y1 Y2 Y3 Y4 Y5
X6 Y6 Y7 Y8 Y9 Y10
.....

CSV FILE
X1 Y1
X2 Y2
X3 Y3
...

Peak Detection

Use Mean and STD to find threshold

Use find_peaks from scipy

Handle overlapping peaks

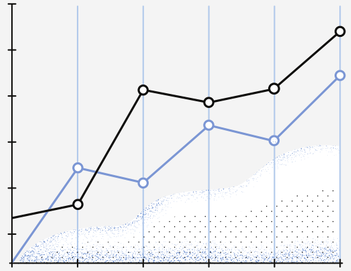
Categorization

Create a dictionary for functional groups & categorize the peaks

Categorize the intensity

Printed Results

Create plots with matplotlib, tables with tabulate, and interactive display with ipywidgets

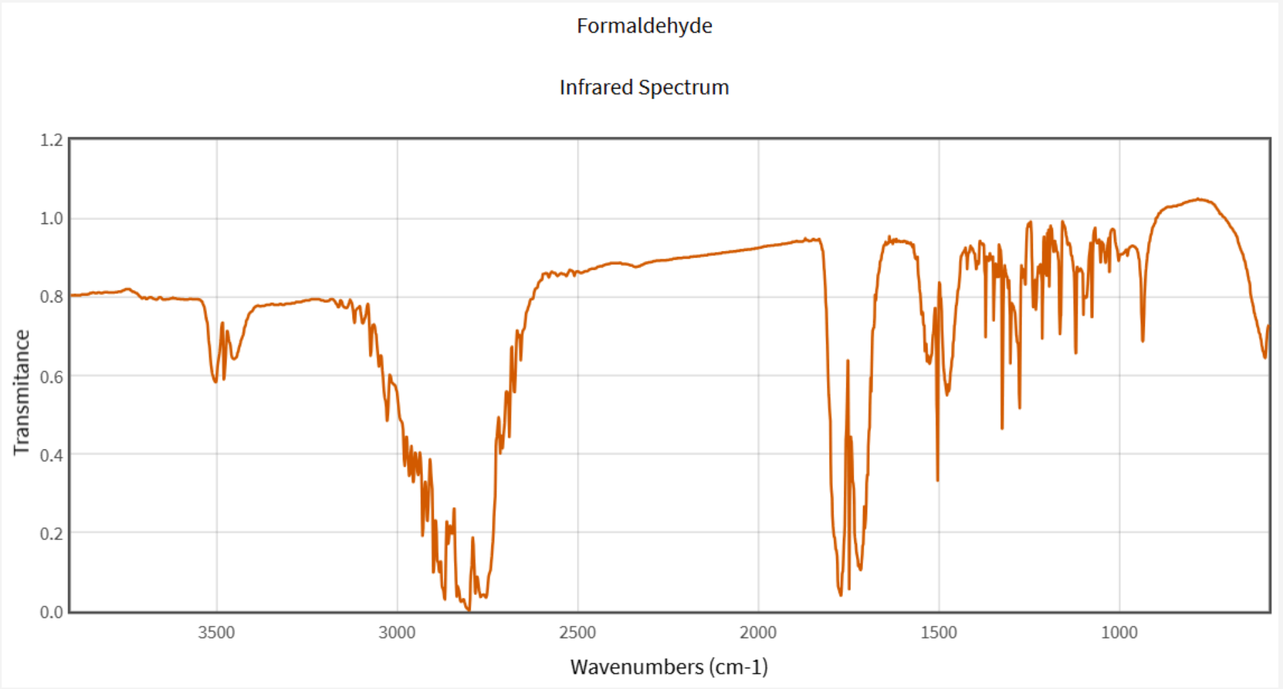


Data Source : NIST Chemistry WebBook, SRD 69

File Conversion

JDX FILE
X1 Y1 Y2 Y3 Y4 Y5
X6 Y6 Y7 Y8 Y9 Y10
.....

CSV FILE
X1 Y1
X2 Y2
X3 Y3
...




```
##MAXY=1.05
##MINY=0.00216665
##NPOINTS=3576
##XYDATA=(X++(Y..Y))
587.635000 0.7290 0.7270 0.7240 0.7200 0.7110
592.271792 0.7060 0.6950 0.6840 0.6681 0.6581
596.908584 0.6460 0.6450 0.6460 0.6470 0.6530
601.545377 0.6570 0.6600 0.6620 0.6630 0.6680
606.182169 0.6740 0.6760 0.6840 0.6910 0.6940
610.818961 0.6970 0.7050 0.7060 0.7070 0.7080
615.455753 0.7120 0.7190 0.7220 0.7240 0.7270
620.092545 0.7310 0.7380 0.7410 0.7420 0.7450
624.729337 0.7490 0.7520 0.7580 0.7580 0.7600
```

```python
import csv
import math
def read_jdx_file(file_path):

#Reads a JCAMP-DX file, extracts the X and Y values, and returns a list of (x, y) pairs.
    with open(file_path, 'r') as file:
        lines = file.readlines()

    x_values = []
    y_values = []
    superX_values=[]
    superY_values=[]

    # Skip all lines that start with '#' and begin processing the data
    data_started = False
    for line in lines:
        line = line.strip()

        # Skip lines that did not start with a number
        if not line or not line[0].isdigit():
            continue

        # If we encounter a non-# line, data starts
        data_started = True

        # Process data lines only
        if data_started and line:
            # Split the line into individual values
            values = list(map(float, line.split()))

            superX_values.append(values[0])  # First value is X
            superY_values.append(values[1:])  # Remaining values are Y
      #result -> superX_values=[x1, x6, ...] superY_values=[(y1,y2,y3,y4,y5), (y6, y7, y8, y9, y10), ...

    for x in superX_values:
      x_values.append(x)
      x_values.append(x+1)
      x_values.append(x+2)
      x_values.append(x+3)
      x_values.append(x+4)
    #result -> x_values=[x1, x2, x3,...]

    for ys in superY_values:
      for y in ys:
        y_values.append(y)
    #result -> y_values=[y1, y2, y3,...]

    if not x_values or not y_values:
        raise ValueError("No data found in the JCAMP-DX file.")
```

Read JDX file

Create empty list for X and Y Values

Skips line with no data

File Conversion

JDX FILE
X1 Y1 Y2 Y3 Y4 Y5
X6 Y6 Y7 Y8 Y9 Y10
.....

CSV FILE
X1 Y1
X2 Y2
X3 Y3
...

Added the missing increament
X1 X5 X10 [superX_values] -> X1 X2 X3... [x_values]

Convert
(Y1,Y2,..,Y5), (Y6,....,Y10),.. [superY_values] -> Y1, Y2, Y3... [y_values]

```python
def generate_xy_pairs(x_values, y_values):
    """
    Converts the list of X values and corresponding list of Y values into (x, y) pairs.
    """
    xy_pairs = []
    counter=0
    while counter < len(y_values):
      y=y_values[counter]
      x=x_values[counter]
      if y==0:
        xy_pairs.append((x, y))
      else:
         y=-math.log(y,10)
         xy_pairs.append((x, y))
      counter+=1
    return xy_pairs
    #result: [(x1, y1), (x2, y2), (x3, y3), ...]

def save_xy_to_csv(output_path, xy_pairs):

#Saves the (x, y) pairs to a CSV file.
    with open(output_path, mode='w', newline='') as file:
        writer = csv.writer(file)

        # Write the header
        writer.writerow(["Wavenumber", "Absorbance"])

        # Write the (x, y) pairs
        for x, y in xy_pairs:
            writer.writerow([x, y])
```

Generate (x,y) pairs from the extracted x_values and y_values

$$A = -log_{10}(T)$$

Save the (x, y) pairs to a CSV file

```python
# Main function
if __name__ == "__main__":
    input_file = "formaldehyde.jdx"  # Replace with the path to your JCAMP-DX file
    output_file = "formaldehyde.csv"  # Output file for (x, y) pairs in CSV format

    try:
        # Read the .jdx file, skipping the first 36 lines
        x_values, y_values = read_jdx_file(input_file)

        # Generate the (x, y) pairs
        xy_pairs = generate_xy_pairs(x_values, y_values)

        # Save the results to a .csv file
        save_xy_to_csv(output_file, xy_pairs)

        print(f"Converted (x, y) data saved to {output_file}")
    except Exception as e:
        print(f"Error: {e}")
```

```
Converted (x, y) data saved to formaldehyde.csv
```

Output: A CSV file with two columns: Wavenumber and Absorbance

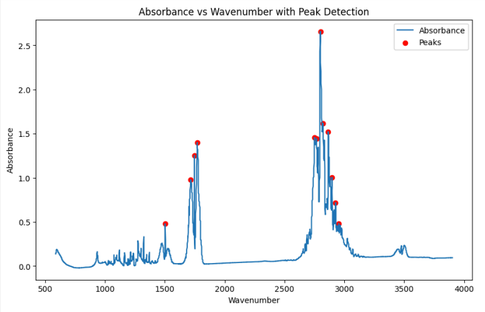
```python
import numpy as np
from ipywidgets import interact, widgets
import pandas as pd
from scipy.signal import find_peaks
import matplotlib.pyplot as plt
from tabulate import tabulate
from IPython.display import display

# Load data
data = pd.read_csv('formaldehyde.csv')

# Check for peaks in 'Absorbance'
# Calculate the mean and standard deviation for threshold
mean_height = data['Absorbance'].mean()
std_height = data['Absorbance'].std()
threshold = mean_height + std_height

# Find peaks above the threshold
peak_indices, properties = find_peaks(data['Absorbance'], height=threshold)

# Loop through all the peaks and check for nearby peaks
unique_peak_indices = []
max_distance = 15  # The distance for overlapping peaks

for i in peak_indices:
    is_unique = True
    for j in unique_peak_indices:
        if abs(data['Wavenumber'][i] - data['Wavenumber'][j]) < max_distance:
            if data['Absorbance'][i] > data['Absorbance'][j]:
                unique_peak_indices.remove(j)
                unique_peak_indices.append(i)
            is_unique = False
            break
    if is_unique:
        unique_peak_indices.append(i)
```

Peak Detection

Use Mean and STD to find threshold

Use find_peaks from scipy

Handle overlapping peaks



Create and design the scatter plot

Printed Results

Categorization

Create a dictionary for functional groups

```python
# Dictionary for functional groups
functional_groups = {
    "C–H Alkanes": (3000, 2850),
    "C–H Alkanes –CH3": [(1450, 1450), (1375, 1375)],
    "C–H Alkanes –CH2–": (1465, 1465),
    "C–H Alkenes": (3100, 3000),
    "C–H Aromatics": [(3150, 3050), (900, 690)],
    "C–H Alkyne": (3300, 3300),
    "C–H Aldehyde": [(2900, 2800), (2800, 2700)],
    "C=C Alkene": (1680, 1600),
    "C=C Aromatic": [(1600, 1600), (1475, 1475)],
    "C≡C Alkyne": (2250, 2100),
    "C=O Aldehyde": (1740, 1720),
    "C=O Ketone": (1725, 1705),
    "C=O Carboxylic acid": [(1725, 1700), (3400, 2400)],
    "C=O Ester": (1750, 1730),
    "C=O Amide": (1670, 1640),
    "C=O Anhydride": [(1810, 1810), (1760, 1760)],
    "C=O Acid chloride": (1800, 1800),
    "C–O Alcohols, Ethers, Esters": (1300, 1000),
    "O–H Alcohols, Phenols (Free)": (3650, 3600),
    "O–H Alcohols, Phenols (H–Bonded)": (3500, 3200),
    "N–H Primary and Secondary Amines": [(3500, 3100), (1640, 1550)],
    "C–N Amines": (1350, 1000),
    "C=N Imines and Oximes": (1690, 1640),
    "C≡N Nitriles": (2260, 2240),
    "X=C=Y Allenes, Ketenes, Isocyanates": (2270, 1950),
    "N=O Nitro (R–NO2)": [(1550, 1550), (1350, 1350)],
    "S–H Mercaptans": (2550, 2550),
    "S=O Sulfoxides": (1050, 1050),
    "C–X Fluoride": (1400, 1000),
    "C–X Chloride": (800, 600),
    "C–X Bromide and Iodide": (667, 0),
}
```

```python
# Create label: 1 if there's a peak, otherwise 0
data['label'] = 0
data.loc[unique_peak_indices, 'label'] = 1

# Visualize data to verify peaks
plt.figure(figsize=(10, 6))
plt.plot(data['Wavenumber'], data['Absorbance'], label='Absorbance')
plt.scatter(data.loc[unique_peak_indices, 'Wavenumber'], data.loc[unique_peak_indices, 'Absorbance'], color='red', label='Peaks')
plt.xlabel('Wavenumber')
plt.ylabel('Absorbance')
plt.title('Absorbance vs Wavenumber with Peak Detection')
plt.legend()
plt.show()
print("\n")
```

```
def combined_analysis(start_wavenumber=None, end_wavenumber=None):
    print("\n")

    # Filter data based on the user-defined range
    if start_wavenumber and end_wavenumber:
        filtered_data = data[(data['Wavenumber'] >= start_wavenumber) & (data['Wavenumber'] <= end_wavenumber)]
        unique_peak_indices_filtered = [i for i in unique_peak_indices if start_wavenumber <= data['Wavenumber'][i] <= end_wavenumber]
    else:
        filtered_data = data
        unique_peak_indices_filtered = unique_peak_indices

    if len(unique_peak_indices_filtered) == 0:
        print("No peaks detected in the selected range. Please select another range.")
        return  # Exit the function if no peaks are found

    peak_values = filtered_data.loc[unique_peak_indices_filtered, ['Wavenumber', 'Absorbance']]
    peak_values['Wavenumber'] = peak_values['Wavenumber'].astype(float)
    peak_values['Absorbance'] = peak_values['Absorbance'].astype(float)

    # Identify the highest peak
    highest_peak_index = peak_values['Absorbance'].idxmax()
    highest_peak_row = peak_values.loc[highest_peak_index]

    # Prepare rows for the combined table
    table_data = []

    # Header row for the table
    table_data.append(["Wavenumber", "Absorbance", "Functional Groups", "Peak Intensity"])

    # Define thresholds for categorization of high, medium, and low
    peak_heights = peak_values['Absorbance'].values
    high_threshold = np.percentile(peak_heights, 80)  # Top 20%
    low_threshold = np.percentile(peak_heights, 20)   # Bottom 20%
```

Make sure that the input from the user does not result in an error

Change the type of the data

Identify the peaks

Create the table for the printed data

Define the threshold for the intensity of the peaks

Preventing out of range error

```python
# Filter data based on the user-defined range
if start_wavenumber and end_wavenumber:
    filtered_data = data[(data['Wavenumber'] >= start_wavenumber) & (data['Wavenumber'] <= end_wavenumber)]
    unique_peak_indices_filtered = [i for i in unique_peak_indices if start_wavenumber <= data['Wavenumber'][i] <= end_wavenumber]
else:
    filtered_data = data
    unique_peak_indices_filtered = unique_peak_indices

if len(unique_peak_indices_filtered) == 0:
    print("No peaks detected in the selected range. Please select another range.")
    return  # Exit the function if no peaks are found
```

```
Default range: Start Wavenumber = 587.63, End Wavenumber = 3902.94
If you wish to analyze a specific range, you can enter the range below:
Start Wave...  500
End Waven...   510

No peaks detected in the selected range. Please select another range.
```

Creating Printed Tables

```python
# Prepare rows for the combined table
table_data = []

# Header row for the table
table_data.append(["Wavenumber", "Absorbance", "Functional Groups", "Peak Intensity"])
```

```
+--------------+-----------------------------------------------------+--------------+
|   Wavenumber | Functional Groups                                   | Absorbance   |
+==============+=====================================================+==============+
```

```python
# Loop through each peak to add functional groups and intensity categorization
for index, row in peak_values.iterrows():
    wavenumber = row['Wavenumber']
    absorbance = row['Absorbance']
    matched_groups = []

    # Find matching functional groups
    for group, value in functional_groups.items():
        if isinstance(value[0], int):  # Single range
            high, low = value
            if low <= wavenumber <= high:
                matched_groups.append(group)
        else:  # Multiple ranges
            for high, low in value:
                if low <= wavenumber <= high:
                    matched_groups.append(group)

    # Categorize peak intensity
    category = "High" if absorbance >= high_threshold else "Low" if absorbance <= low_threshold else "Medium"

    # Style the highest peak row
    if index == highest_peak_index:
        wavenumber = f"\033[1m\033[31m{wavenumber:.2f}\033[0m"  # Bold + Red
        category = f"\033[1m\033[31m{category}\033[0m"          # Bold + Red

    # Add row to the table
    functional_groups_str = ', '.join(matched_groups) if matched_groups else "No matching functional group"
    table_data.append([wavenumber, functional_groups_str, category])
```

Create a for loop for repetition

Check if the peak is in the range of the functional group

Categorize the intensity of each peak

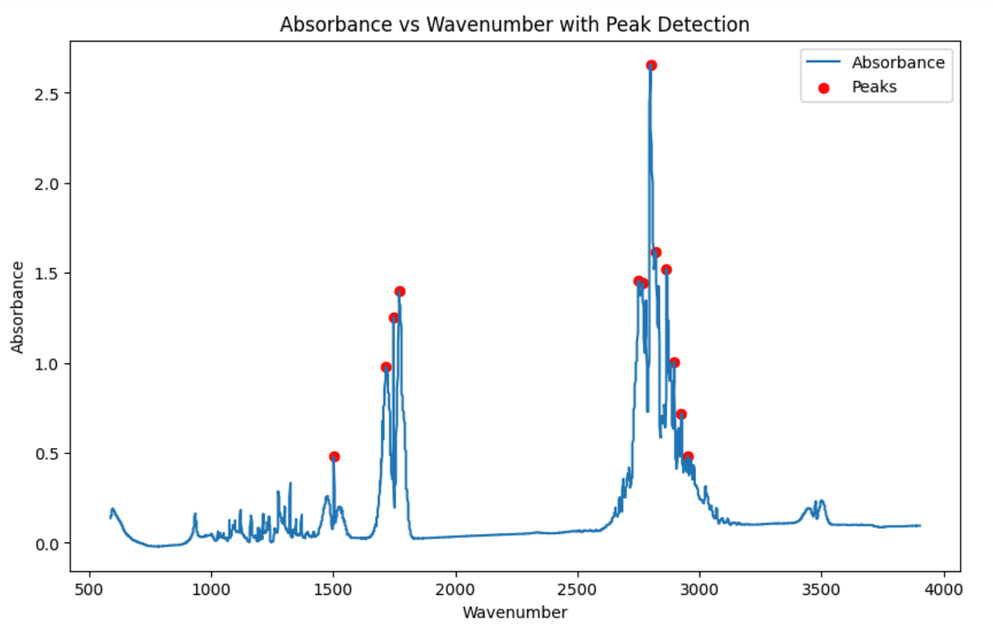
Label the highest peak in red

Print out the result

Categorizing the Intensity

```
# Define thresholds for categorization of high, medium, and low
peak_heights = peak_values['Absorbance'].values
high_threshold = np.percentile(peak_heights, 80)  # Top 20%
low_threshold = np.percentile(peak_heights, 20)   # Bottom 20%
```

```
# Loop through each peak to add functional groups and intensity categorization
for index, row in peak_values.iterrows():
    # Categorize peak intensity
    category = "High" if absorbance >= high_threshold else "Low" if absorbance <= low_threshold else "Medium"
```



80%

20%

Matching the Peaks to the Functional Groups

```python
for index, row in peak_values.iterrows():
    wavenumber = row['Wavenumber']
    absorbance = row['Absorbance']
    matched_groups = []

    # Find matching functional groups
    for group, value in functional_groups.items():
        if isinstance(value[0], int):  # Single range
            high, low = value
            if low <= wavenumber <= high:
                matched_groups.append(group)
        else:  # Multiple ranges
            for high, low in value:
                if low <= wavenumber <= high:
                    matched_groups.append(group)
```

Functional group	Mode of vibration		Wavenumber (cm^{-1})	Intensity
C–H	Alkanes	(stretching)	3000-2850	Strong
	$-CH_3$	(bending)	1450 and 1375	Medium
	$-CH_2-$	(bending)	1465	Medium
	Alkenes	(stretching)	3100-3000	Medium
		(out of plane bending)	1000-650	Strong
	Aromatics	(stretching)	3150-3050	Strong
		(out of plane bending)	900-690	Strong
	Alkyne	(stretching)	ca. 3300	Strong
	Aldehyde		2900-2800	Weak
			2800-2700	Weak
C=C	Alkane	not interpretative useful		
	Alkene		1680-1600	Medium to Weak
	Aromatic		1600 and 1475	Medium to Weak
C≡C	Alkyne		2250-2100	Medium to Weak

```python
# Dictionary for functional groups
functional_groups = {
    "C–H Alkanes": (3000, 2850),
    "C–H Alkanes –CH3": [(1450, 1450), (1375, 1375)],
    "C–H Alkanes –CH2–": (1465, 1465),
    "C–H Alkenes": (3100, 3000),
    "C–H Aromatics": [(3150, 3050), (900, 690)],
    "C–H Alkyne": (3300, 3300),
    "C–H Aldehyde": [(2900, 2800), (2800, 2700)],
    "C=C Alkene": (1680, 1600),
    "C=C Aromatic": [(1600, 1600), (1475, 1475)],
    "C≡C Alkyne": (2250, 2100),
    "C=O Aldehyde": (1740, 1720),
    "C=O Ketone": (1725, 1705),
    "C=O Carboxylic acid": [(1725, 1700), (3400, 2400)],
    "C=O Ester": (1750, 1730),
    "C=O Amide": (1670, 1640),
    "C=O Anhydride": [(1810, 1810), (1760, 1760)],
    "C=O Acid chloride": (1800, 1800),
    "C–O Alcohols, Ethers, Esters": (1300, 1000),
    "O–H Alcohols, Phenols (Free)": (3650, 3600),
    "O–H Alcohols, Phenols (H-Bonded)": (3500, 3200),
    "N–H Primary and Secondary Amines": [(3500, 3100), (1640, 1550)],
    "C–N Amines": (1350, 1000),
    "C=N Imines and Oximes": (1690, 1640),
    "C≡N Nitriles": (2260, 2240),
    "X=C=Y Allenes, Ketenes, Isocyanates": (2270, 1950),
    "N=O Nitro (R–NO2)": [(1550, 1550), (1350, 1350)],
    "S–H Mercaptans": (2550, 2550),
    "S=O Sulfoxides": (1050, 1050),
    "C–X Fluoride": (1400, 1000),
    "C–X Chloride": (800, 600),
    "C–X Bromide and Iodide": (667, 0),
}
```

Adding the Values to the Table

```
        # Style the highest peak row
        if index == highest_peak_index:
            wavenumber = f"\033[1m\033[31m{wavenumber:.2f}\033[0m"  # Bold + Red
            category = f"\033[1m\033[31m{category}\033[0m"          # Bold + Red

        # Add row to the table
        functional_groups_str = ', '.join(matched_groups) if matched_groups else "No matching functional group"
        table_data.append([wavenumber, functional_groups_str, category])
```

```
# Print the combined table
print(tabulate(table_data, headers="firstrow", tablefmt="grid"))
print("\n")
```

Wavenumber	Functional Groups	Absorbance
2752.38	C–H Aldehyde, C=O Carboxylic acid	Medium
2768.93	C–H Aldehyde, C=O Carboxylic acid	Medium
2800.38	C–H Aldehyde, C=O Carboxylic acid	**High**
2822.57	C–H Aldehyde, C=O Carboxylic acid	High
2867.3	C–H Alkanes, C–H Aldehyde, C=O Carboxylic acid	Medium
2898.76	C–H Alkanes, C–H Aldehyde, C=O Carboxylic acid	Medium
2928.58	C–H Alkanes, C=O Carboxylic acid	Low
2954.4	C–H Alkanes, C=O Carboxylic acid	Low

Printed Results

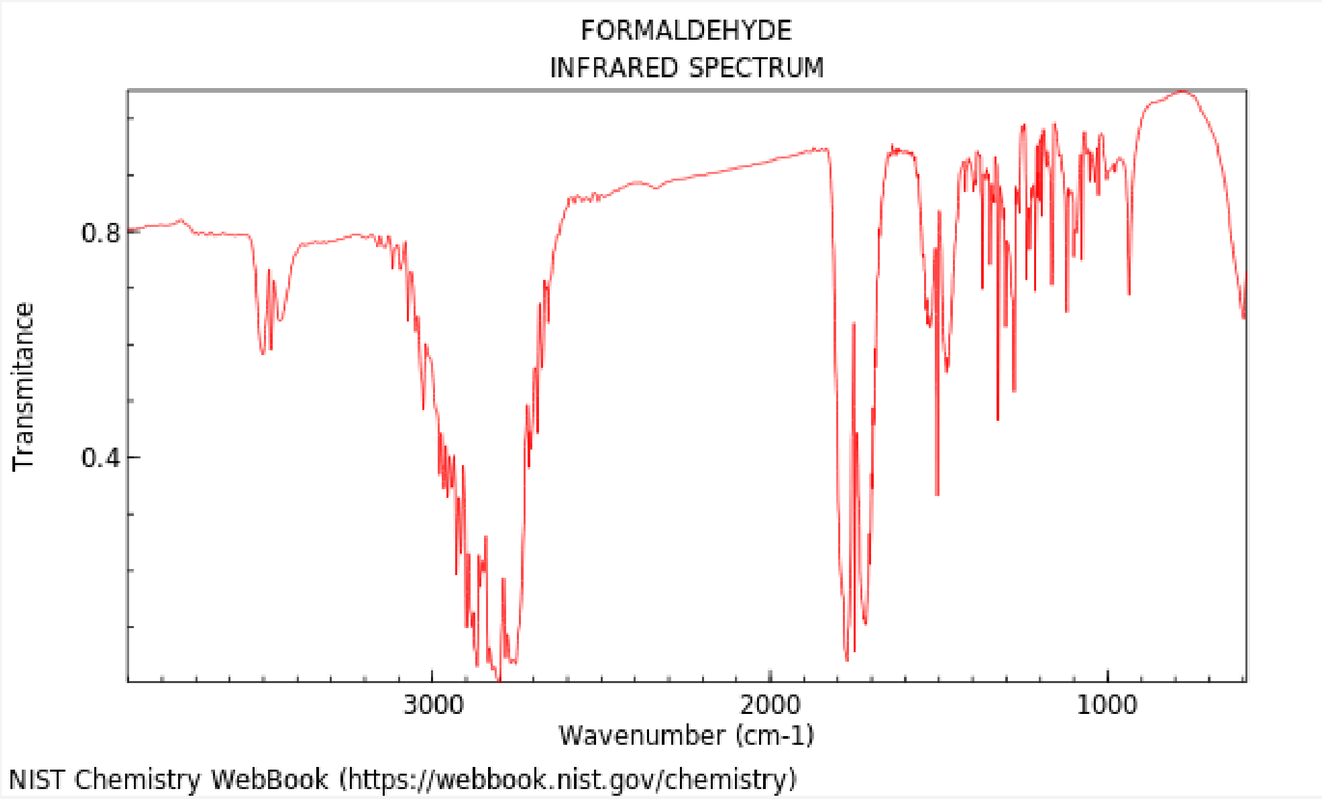
Let users input the range of the wavenumber

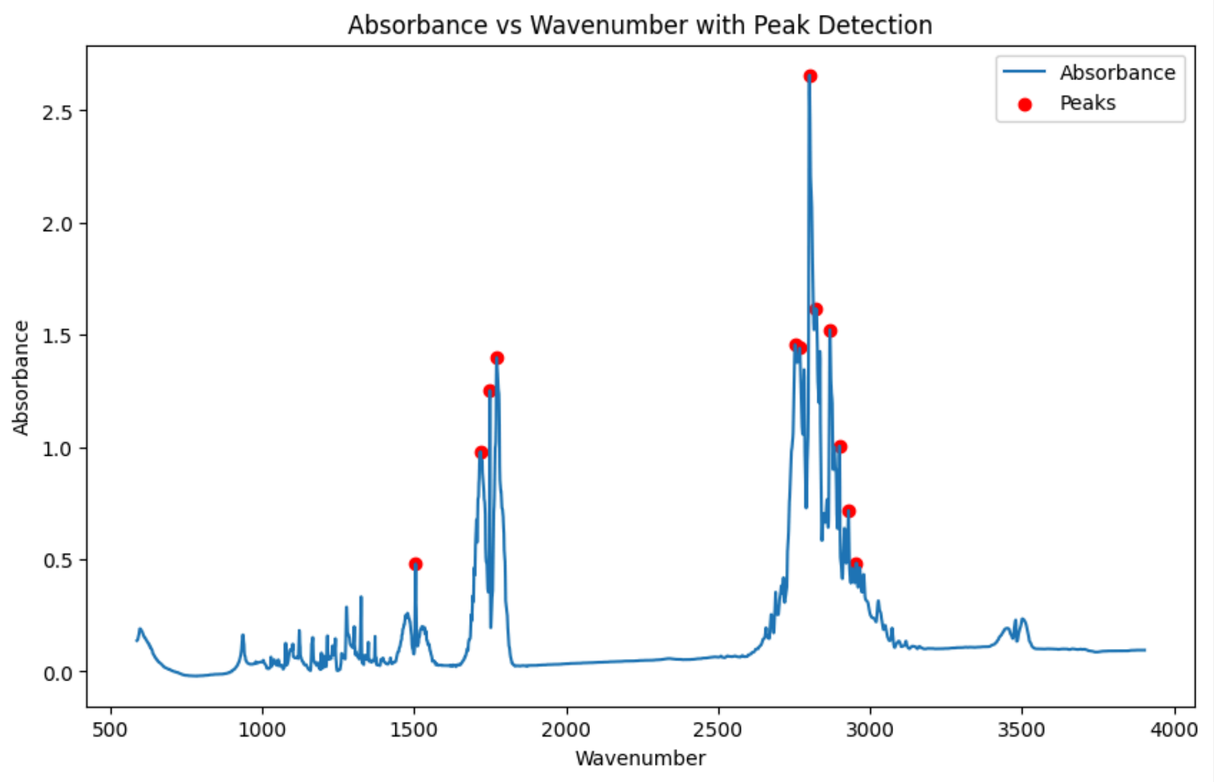
```
Default range: Start Wavenumber = 587.63, End Wavenumber = 3902.94
If you wish to analyze a specific range, you can enter the range below:
Start Wave...  2500
End Waven...   3000
```

```python
# Interactive analysis for a user-defined range
def interactive_analysis():
    interact(
        analyze_range,
        start=widgets.FloatText(value=data['Wavenumber'].min(), description='Start Wavenumber:'),
        end=widgets.FloatText(value=data['Wavenumber'].max(), description='End Wavenumber:')
    )

# Function to analyze the selected range
def analyze_range(start, end):
    combined_analysis(start, end)

# Option to analyze a specific range
print(f"Default range: Start Wavenumber = {data['Wavenumber'].min():.2f}, End Wavenumber = {data['Wavenumber'].max():.2f}")
print("If you wish to analyze a specific range, you can enter the range below:")
interactive_analysis()
```

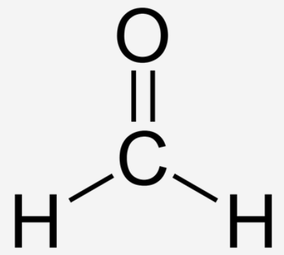
RESULT

Formaldehyde

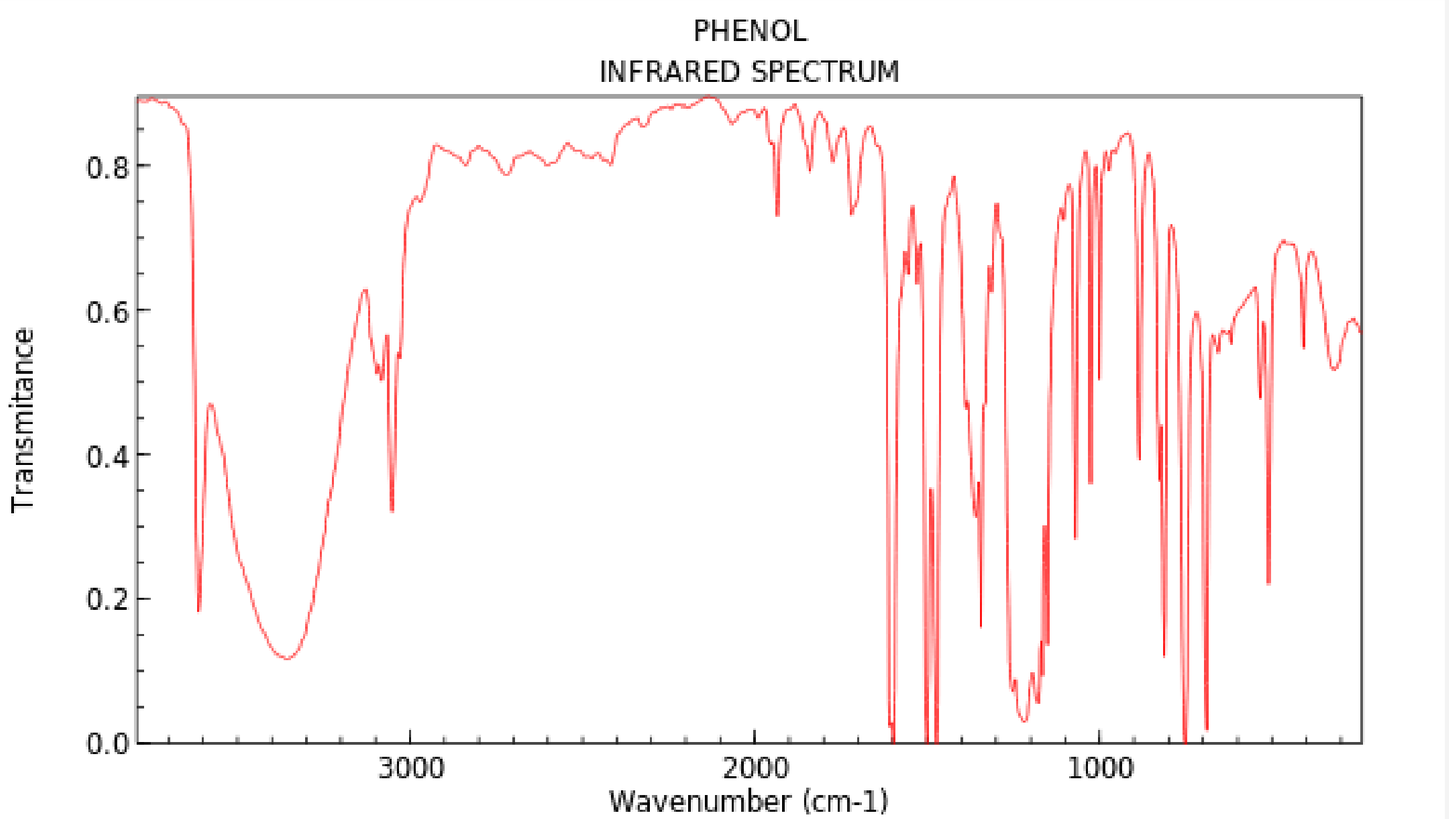


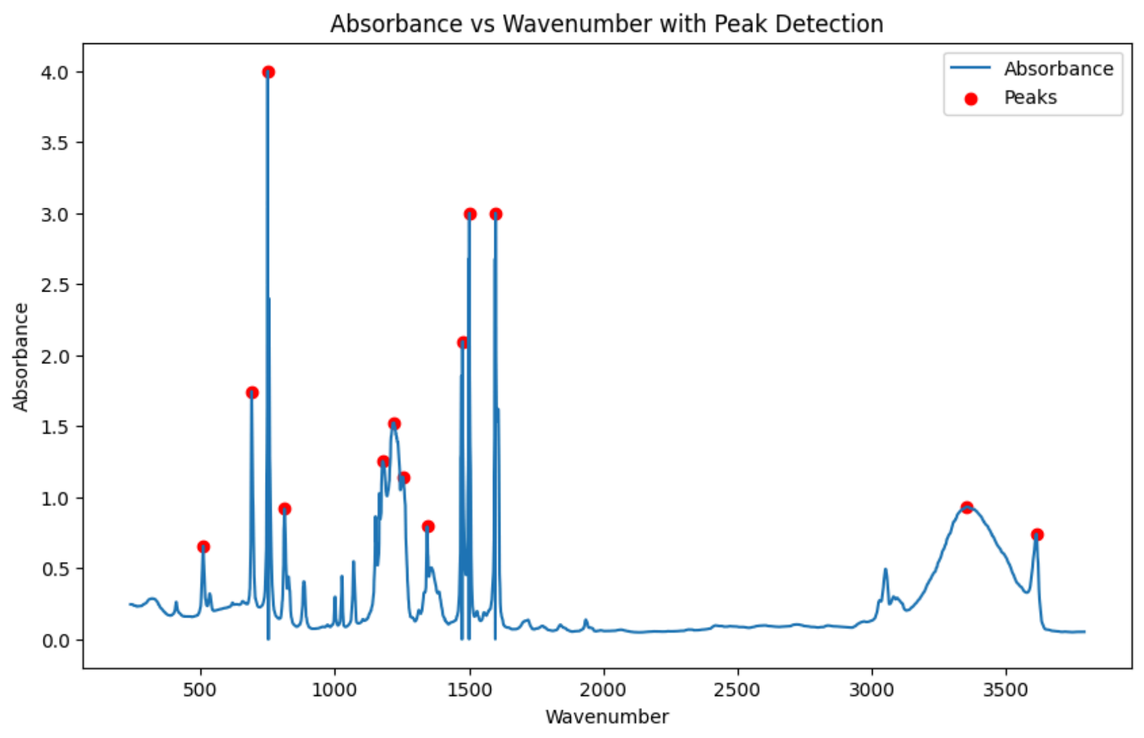


Wavenumber	Functional Groups	Absorbance
1505.08	No matching functional group	Low
1717.38	C=O Ketone, C=O Carboxylic acid	Medium
1748.83	C=O Ester	Medium
1771.02	No matching functional group	Medium
2752.38	C–H Aldehyde, C=O Carboxylic acid	Medium
2768.93	C–H Aldehyde, C=O Carboxylic acid	Medium
2800.38	C–H Aldehyde, C=O Carboxylic acid	**High**
2822.57	C–H Aldehyde, C=O Carboxylic acid	High
2867.3	C–H Alkanes, C–H Aldehyde, C=O Carboxylic acid	High
2898.76	C–H Alkanes, C–H Aldehyde, C=O Carboxylic acid	Medium
2928.58	C–H Alkanes, C=O Carboxylic acid	Low
2954.4	C–H Alkanes, C=O Carboxylic acid	Low

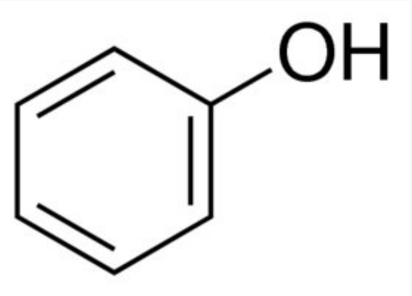


Phenol

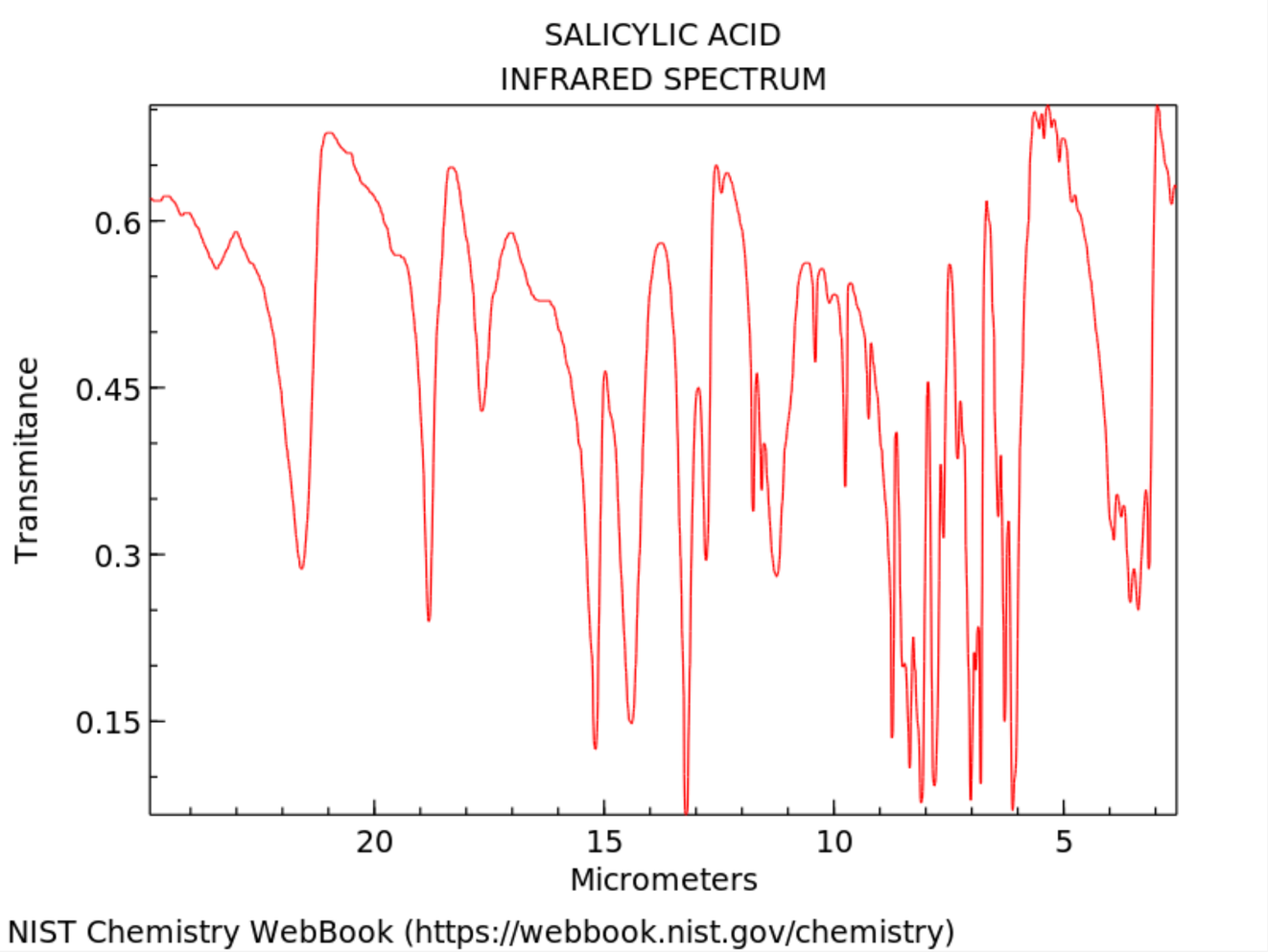

PHENOL
INFRARED SPECTRUM
Transmitance
0.8
0.6
0.4
0.2
0.0
3000
2000
1000
Wavenumber (cm-1)
NIST Chemistry WebBook (https://webbook.nist.gov/chemistry)



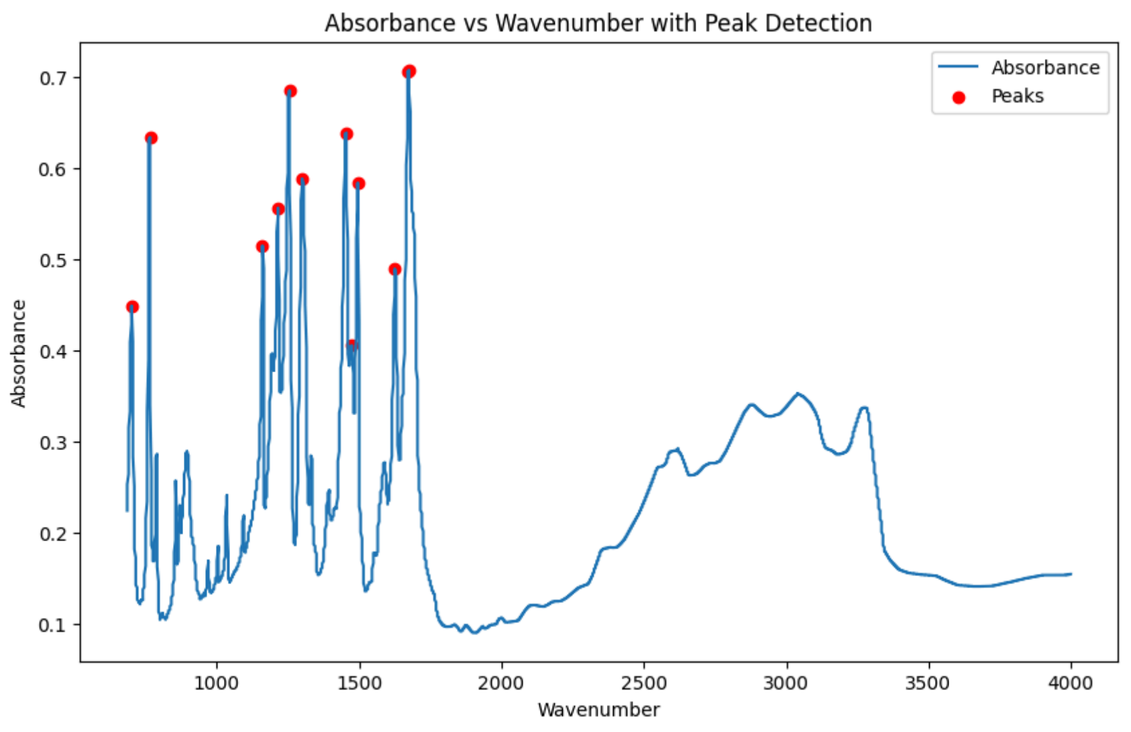
Wavenumber	Functional Groups	Absorbance
508.768	C–X Bromide and Iodide	Low
689.234	C–X Chloride	Medium
749.41	C–H Aromatics, C–X Chloride	**High**
811.522	C–H Aromatics	Medium
1177.45	C–O Alcohols, Ethers, Esters, C–N Amines, C–X Fluoride	Medium
1217.91	C–O Alcohols, Ethers, Esters, C–N Amines, C–X Fluoride	Medium
1252.43	C–O Alcohols, Ethers, Esters, C–N Amines, C–X Fluoride	Medium
1344.13	C–N Amines, C–X Fluoride	Low
1475.34	No matching functional group	Medium
1500.93	No matching functional group	High
1598.63	N–H Primary and Secondary Amines	High
3355.18	C=O Carboxylic acid, O–H Alcohols, Phenols (H-Bonded), N–H Primary and Secondary Amines	Medium
3613.62	O–H Alcohols, Phenols (Free)	Low



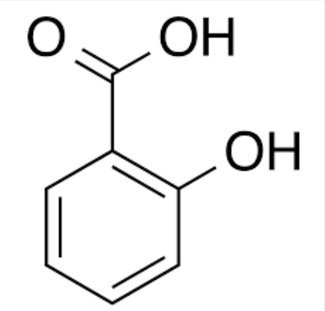
Salicylic Acid



NIST Chemistry WebBook (https://webbook.nist.gov/chemistry)



Wavenumber	Functional Groups	Absorbance
3277	C=O Carboxylic acid, O–H Alcohols, Phenols (H-Bonded), N–H Primary and Secondary Amines	Low
3042	C–H Alkenes, C=O Carboxylic acid	Low
2881	C–H Alkanes, C–H Aldehyde, C=O Carboxylic acid	Low
1679	C=C Alkene, C=N Imines and Oximes	**High**
1625	C=C Alkene, N–H Primary and Secondary Amines	Medium
1496	No matching functional group	Medium
1479	No matching functional group	Medium
1458	No matching functional group	High
1302	C–N Amines, C–X Fluoride	Medium
1258	C–O Alcohols, Ethers, Esters, C–N Amines, C–X Fluoride	High
1217	C–O Alcohols, Ethers, Esters, C–N Amines, C–X Fluoride	Medium
1199	C–O Alcohols, Ethers, Esters, C–N Amines, C–X Fluoride	Medium
1160	C–O Alcohols, Ethers, Esters, C–N Amines, C–X Fluoride	Medium
769	C–H Aromatics, C–X Chloride	Medium
702	C–H Aromatics, C–X Chloride	Medium



Group Work

Charoline 黃菊花
B10611026

Made the plot and finding the peaks

Asia 黃夏彤
B11502120

Adding peak description and user interface

Teresa 陳翠珍
B11504086

Converting JDX file to excel file

Thalia 郭伶美
B11B02043

Code fixing, evaluating, and result testing


THANK YOU!