Problem_2

September 15, 2025

[8]:
```
pip install pandas scipy seaborn
```

```
Requirement already satisfied: pandas in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (2.1.4)
Requirement already satisfied: scipy in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (1.11.4)
Collecting seaborn
  Using cached seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Requirement already satisfied: numpy<2,>=1.23.2 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
pandas) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
pandas) (2023.3.post1)
Requirement already satisfied: tzdata>=2022.1 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
pandas) (2023.3)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
seaborn) (3.8.3)
Requirement already satisfied: contourpy>=1.0.1 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.2.0)
Requirement already satisfied: cycler>=0.10 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (4.50.0)
Requirement already satisfied: kiwisolver>=1.3.1 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (24.2)
```

```
Requirement already satisfied: pillow>=8 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (10.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
matplotlib!=3.6.1,>=3.4->seaborn) (3.1.2)
Requirement already satisfied: six>=1.5 in
/Users/varagantibasanthkumar/miniconda3/lib/python3.11/site-packages (from
python-dateutil>=2.8.2->pandas) (1.16.0)
Using cached seaborn-0.13.2-py3-none-any.whl (294 kB)
Installing collected packages: seaborn
Successfully installed seaborn-0.13.2
Note: you may need to restart the kernel to use updated packages.
```

[9]:
```python
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from scipy import stats
import warnings
warnings.filterwarnings('ignore')
```

[10]:
```python
def load_sentinel2_data(file_path='/Users/varagantibasanthkumar/Desktop/Remote␣
 ↪sensing - IMGS 589/Homework1/sentinel2_rochester.npy'):
    data = np.load(file_path)
    print("Data loaded successfully!")
    print("Shape: " + str(data.shape))
    return data
```

[11]:
```python
def get_band_info():
    bands = {
        0: 'B1', 1: 'B2', 2: 'B3', 3: 'B4', 4: 'B5', 5: 'B6',
        6: 'B7', 7: 'B8', 8: 'B8A', 9: 'B9', 10: 'B11', 11: 'B12'
    }
    return bands
```

[12]:
```python
def find_no_data(data, threshold=0.001):
    mask = (data < threshold) | np.isnan(data)
    return mask
```

[13]:
```python
def calculate_stats(data):
    print("Calculating band statistics...")

    no_data_mask = find_no_data(data)
    band_names = get_band_info()
    results = {}

    for i in range(12):
```

```
        band_name = band_names[i]
        band_data = data[:, :, i]

        # get valid pixels only
        valid_pixels = band_data[~no_data_mask[:, :, i]]

        if len(valid_pixels) == 0:
            continue

        # basic stats
        mean_val = np.mean(valid_pixels)
        std_val = np.std(valid_pixels)
        min_val = np.min(valid_pixels)
        max_val = np.max(valid_pixels)

        # quartiles
        q1 = np.percentile(valid_pixels, 25)
        median = np.median(valid_pixels)
        q3 = np.percentile(valid_pixels, 75)

        # skewness and kurtosis
        skew = stats.skew(valid_pixels)
        kurt = stats.kurtosis(valid_pixels)

        results[band_name] = {
            'mean': mean_val,
            'std': std_val,
            'min': min_val,
            'max': max_val,
            'q1': q1,
            'median': median,
            'q3': q3,
            'skewness': skew,
            'kurtosis': kurt,
            'count': len(valid_pixels)
        }

        print(band_name + " done")

    return results
```

[14]:

```
def explain_stats():
    print("\nWhat these statistics mean:")
    print("Mean: average value")
    print("Std: how spread out the data is")
    print("Min/Max: smallest and largest values")
    print("Q1, Median, Q3: quartiles (25%, 50%, 75%)")
```

```
    print("Skewness: if data is symmetric or not")
    print("Kurtosis: how \"heavy\" the tails are")
```

[15]:
```
def make_table(stats_dict):
    print("\nBand Statistics Table:")

    # make a simple table
    table_data = []
    for band, values in stats_dict.items():
        row = {
            'Band': band,
            'Mean': round(values['mean'], 4),
            'Std': round(values['std'], 4),
            'Min': round(values['min'], 4),
            'Max': round(values['max'], 4),
            'Q1': round(values['q1'], 4),
            'Median': round(values['median'], 4),
            'Q3': round(values['q3'], 4),
            'Skewness': round(values['skewness'], 4),
            'Kurtosis': round(values['kurtosis'], 4)
        }
        table_data.append(row)

    df = pd.DataFrame(table_data)
    print(df)
    return df
```

[16]:
```
def standardize_data(data):
    print("\nStandardizing data...")

    no_data_mask = find_no_data(data)
    standardized = np.full_like(data, np.nan)

    for i in range(12):
        band_data = data[:, :, i]
        mask = no_data_mask[:, :, i]

        # get valid data
        valid_data = band_data[~mask]

        if len(valid_data) == 0:
            continue

        # calculate mean and std
        mean_val = np.mean(valid_data)
        std_val = np.std(valid_data)
```

```
        # z-score formula
        z_scores = (band_data - mean_val) / std_val

        # keep no-data as NaN
        z_scores[mask] = np.nan

        standardized[:, :, i] = z_scores

        print("Band " + str(i+1) + " standardized")

    return standardized
```

[17]:
```
def explain_standardization():
    print("\nWhat standardization does:")
    print("- Makes mean = 0 and std = 1 for each band")
    print("- Allows comparison between different bands")
    print("- Helps find outliers (values > 3 or < -3)")
    print("- Common preprocessing step")
```

[18]:
```
def plot_histograms(original_data, standardized_data):
    print("\nMaking histograms...")

    band_names = get_band_info()
    no_data_mask = find_no_data(original_data)

    fig, axes = plt.subplots(4, 3, figsize=(15, 20))
    fig.suptitle('Band Histograms with Outliers', fontsize=16)

    axes = axes.flatten()

    for i in range(12):
        ax = axes[i]
        band_name = band_names[i]

        # get data
        orig_band = original_data[:, :, i]
        std_band = standardized_data[:, :, i]
        mask = no_data_mask[:, :, i]

        # valid pixels only
        valid_orig = orig_band[~mask]
        valid_std = std_band[~mask]

        if len(valid_orig) == 0:
            continue

        # plot histogram
```

```
        ax.hist(valid_orig, bins=50, alpha=0.7, color='blue')

        # find outliers (> 3 std devs)
        outliers = np.abs(valid_std) > 3
        outlier_values = valid_orig[outliers]

        # plot outliers as red dots
        if len(outlier_values) > 0:
            y_pos = np.random.uniform(0, ax.get_ylim()[1]*0.8,␣
↪len(outlier_values))
            ax.scatter(outlier_values, y_pos, color='red', s=10, alpha=0.7)

        # add title
        mean_val = np.mean(valid_orig)
        std_val = np.std(valid_orig)
        outlier_pct = (len(outlier_values) / len(valid_orig)) * 100

        title = band_name + "\nMean: " + str(round(mean_val, 3)) + ", Std: " +␣
↪str(round(std_val, 3)) + "\nOutliers: " + str(round(outlier_pct, 1)) + "%"
        ax.set_title(title, fontsize=9)
        ax.set_xlabel('Reflectance')
        ax.set_ylabel('Count')

    plt.tight_layout()
    plt.savefig('histograms.png', dpi=150, bbox_inches='tight')
    plt.show()
    print("Histograms saved as histograms.png")
```

[19]:
```
def analyze_outliers(standardized_data):
    print("\nOutlier Analysis:")

    band_names = get_band_info()
    no_data_mask = find_no_data(standardized_data)

    print("Band\tTotal\t>2 \t>3 \t%>2 \t%>3 ")
    print("-" * 40)

    for i in range(12):
        band_name = band_names[i]
        std_band = standardized_data[:, :, i]
        mask = no_data_mask[:, :, i]

        valid_data = std_band[~mask]

        if len(valid_data) == 0:
            continue
```

```
        # count outliers
        outliers_2 = np.sum(np.abs(valid_data) > 2)
        outliers_3 = np.sum(np.abs(valid_data) > 3)

        pct_2 = (outliers_2 / len(valid_data)) * 100
        pct_3 = (outliers_3 / len(valid_data)) * 100

        print(band_name + "\t" + str(len(valid_data)) + "\t" + str(outliers_2)␣
 ↪+ "\t" + str(outliers_3) + "\t" + str(round(pct_2, 1)) + "\t" +␣
 ↪str(round(pct_3, 1)))
```

[20]:
```
def main():
    print("Problem 2: Band Statistics and Standardization")
    print("=" * 50)

    # load data
    data = load_sentinel2_data()

    # part a: calculate statistics
    stats = calculate_stats(data)
    explain_stats()
    table = make_table(stats)

    # part b: standardize data
    standardized = standardize_data(data)
    explain_standardization()

    # plot histograms
    plot_histograms(data, standardized)

    # analyze outliers
    analyze_outliers(standardized)

    print("\nDone with the analysis..!!!!!")

if __name__ == "__main__":
    main()
```

```
Problem 2: Band Statistics and Standardization
==================================================
Data loaded successfully!
Shape: (954, 716, 12)
Calculating band statistics…
B1 done
B2 done
B3 done
B4 done
B5 done
```

B6 done
B7 done
B8 done
B8A done
B9 done
B11 done
B12 done

What these statistics mean:
Mean: average value
Std: how spread out the data is
Min/Max: smallest and largest values
Q1, Median, Q3: quartiles (25%, 50%, 75%)
Skewness: if data is symmetric or not
Kurtosis: how "heavy" the tails are

Band Statistics Table:

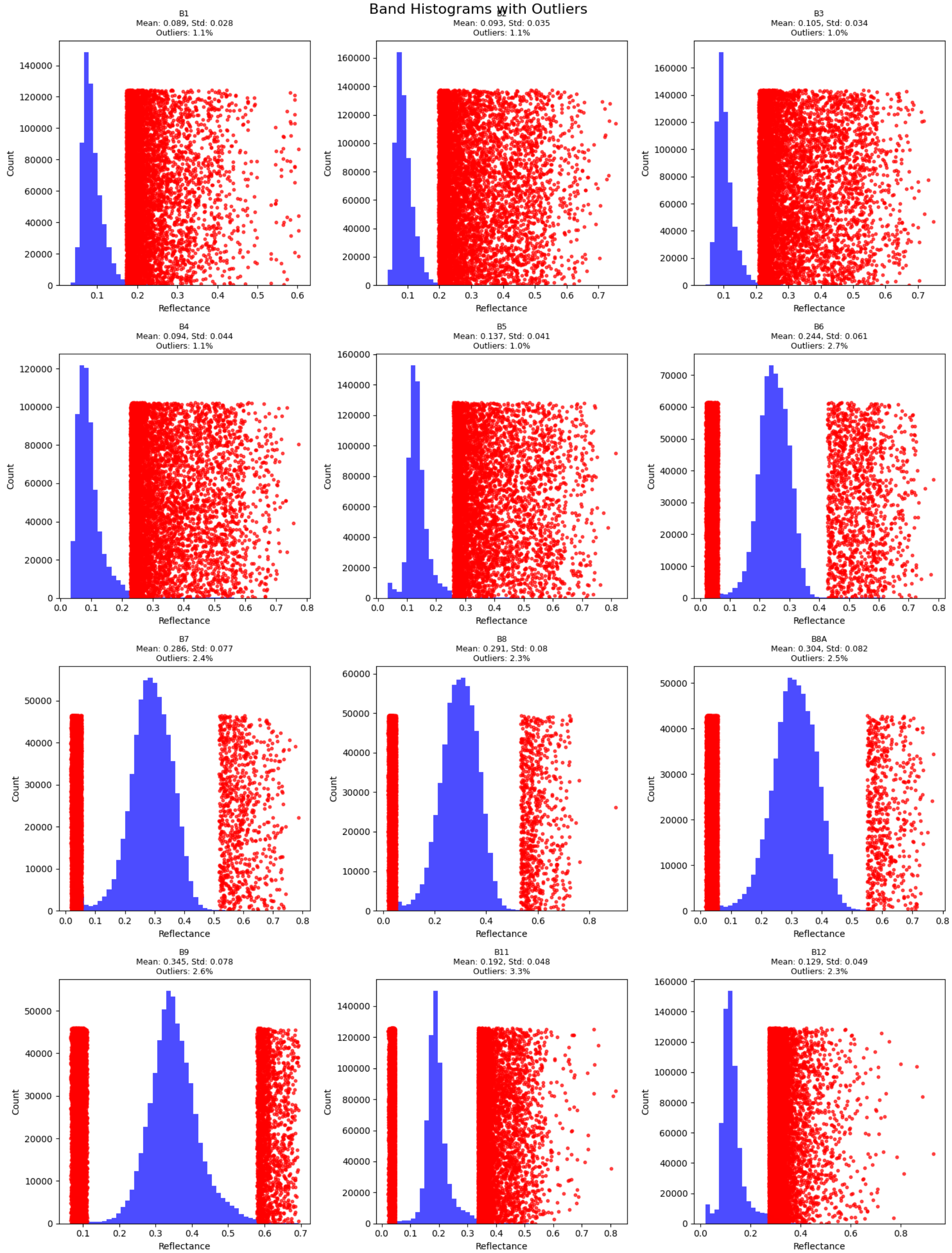
	Band	Mean	Std	Min	Max	Q1	Median	Q3	Skewness	Kurtosis
0	B1	0.0887	0.0279	0.0333	0.6021	0.0709	0.0829	0.1007	2.9205	23.7974
1	B2	0.0925	0.0350	0.0386	0.7542	0.0716	0.0853	0.1053	4.4789	43.1080
2	B3	0.1055	0.0344	0.0430	0.7484	0.0858	0.0987	0.1168	4.6922	46.2931
3	B4	0.0943	0.0445	0.0326	0.7728	0.0663	0.0850	0.1093	3.2314	23.4076
4	B5	0.1367	0.0409	0.0346	0.8159	0.1152	0.1310	0.1506	3.1080	27.3514
5	B6	0.2436	0.0610	0.0176	0.7830	0.2131	0.2472	0.2830	-0.7085	3.4061
6	B7	0.2858	0.0773	0.0161	0.7859	0.2444	0.2903	0.3377	-0.7101	1.7201
7	B8	0.2914	0.0803	0.0188	0.9030	0.2475	0.2976	0.3464	-0.7161	1.4222
8	B8A	0.3035	0.0822	0.0157	0.7689	0.2605	0.3098	0.3595	-0.8215	1.6743
9	B9	0.3451	0.0779	0.0653	0.6934	0.3052	0.3446	0.3882	-0.3957	2.2100
10	B11	0.1917	0.0484	0.0218	0.8196	0.1710	0.1889	0.2086	0.5219	6.4363
11	B12	0.1292	0.0486	0.0205	0.9295	0.1020	0.1204	0.1427	1.9499	8.0381

Standardizing data…
Band 1 standardized
Band 2 standardized

```
Band 3 standardized
Band 4 standardized
Band 5 standardized
Band 6 standardized
Band 7 standardized
Band 8 standardized
Band 9 standardized
Band 10 standardized
Band 11 standardized
Band 12 standardized

What standardization does:
- Makes mean = 0 and std = 1 for each band
- Allows comparison between different bands
- Helps find outliers (values > 3 or < -3)
- Common preprocessing step

Making histograms…
```




```
Histograms saved as histograms.png

Outlier Analysis:
```

```
Band    Total   >2      >3      %>2     %>3
----------------------------------------
B1      250040  20697   7092    8.3     2.8
B2      247924  15450   6636    6.2     2.7
B3      243315  16604   6573    6.8     2.7
B4      243780  25517   6903    10.5    2.8
B5      258693  21809   6261    8.4     2.4
B6      331840  4027    1341    1.2     0.4
B7      330408  4653    841     1.4     0.3
B8      335393  4392    756     1.3     0.2
B8A     336217  4034    611     1.2     0.2
B9      312530  17551   2080    5.6     0.7
B11     288172  24177   6869    8.4     2.4
B12     242053  33001   14252   13.6    5.9

Done!
```

[ ]: