```
import pandas as pd
import numpy as np
import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt
```

```
import pandas as pd
dtype = {
    "x": "float64",
    "y": "float64",
    "z": "float64",
    "intensity": "float64"
}

df = pd.read_csv('LAS_18258650.las.csv', header=None, delim_whitespace=True,
names=["x", "y", "z", "intensity"], on_bad_lines='skip')

df["x"] = pd.to_numeric(df["x"], errors='coerce')
df["y"] = pd.to_numeric(df["y"], errors='coerce')
df["z"] = pd.to_numeric(df["z"], errors='coerce')
df["intensity"] = pd.to_numeric(df["intensity"], errors='coerce')
```

```
/var/folders/r5/zf_7_mvs0z75m2t2zs7q4znr0000gn/T/ipykernel_67871/1292972071.py:10:
DtypeWarning: Columns (0,1,2,3) have mixed types. Specify dtype option on import or
set low_memory=False.
  df = pd.read_csv('LAS_18258650.las.csv', header=None, delim_whitespace=True,
names=["x", "y", "z", "intensity"], on_bad_lines='skip')
```

```
df = df.dropna().reset_index(drop=True)
df
```

	x	y	z	intensity
0	1182513.0	1867463.0	592.0	36578.0
1	1182518.0	1867473.0	592.0	15539.0
2	1182511.0	1867460.0	592.0	38539.0
3	1182508.0	1867465.0	592.0	32019.0
4	1182515.0	1867456.0	630.0	11671.0
...	...	...	...	...
17045629	1184990.0	1865011.0	604.0	5080.0
17045630	1184994.0	1865011.0	608.0	9046.0
17045631	1184998.0	1865012.0	612.0	4459.0
17045632	1184998.0	1865014.0	616.0	14038.0
17045633	1184995.0	1865001.0	618.0	10085.0

17045634 rows × 4 columns

1. Extract a random subset of 1 million rows from the dataset using NumPy. Calculate the max□imum, minimum, mean, and standard deviation for each column (x, y, z, intensity) in both the original dataset and the 1-million-row subset. Present these statistics in a comparative table.

```
subset_df = df.sample(n=1000000, random_state=42)
```

```
original_stats = {
    'max': df.max(),
    'min': df.min(),
    'mean': df.mean(),
    'std': df.std()
}


subset_stats = {
    'max': subset_df.max(),
    'min': subset_df.min(),
    'mean': subset_df.mean(),
    'std': subset_df.std()
}


stats_df = pd.DataFrame({
    'Original Max': original_stats['max'],
    'Subset Max': subset_stats['max'],
    'Original Min': original_stats['min'],
    'Subset Min': subset_stats['min'],
    'Original Mean': original_stats['mean'],
    'Subset Mean': subset_stats['mean'],
    'Original Std': original_stats['std'],
    'Subset Std': subset_stats['std']
})

stats_df
```

	Original Max	Subset Max	Original Min	Subset Min	Original Mean	Subset Mean	Original Std	Subset Std
x	1185000.0	1185000.0	1182500.0	1182500.0	1.183805e+06	1.183806e+06	703.109090	703.093925
y	1867500.0	1867499.0	1865000.0	1865000.0	1.866243e+06	1.866244e+06	697.719147	698.090008
z	853.0	853.0	568.0	568.0	6.230946e+02	6.231230e+02	34.555547	34.597015
intensity	61143.0	59174.0	0.0	0.0	1.492827e+04	1.491416e+04	13791.858399	13778.485843

2. Compute the correlation coefficients between every pair of columns. Identify if there are any significant (in the non-technical sense) correlations.

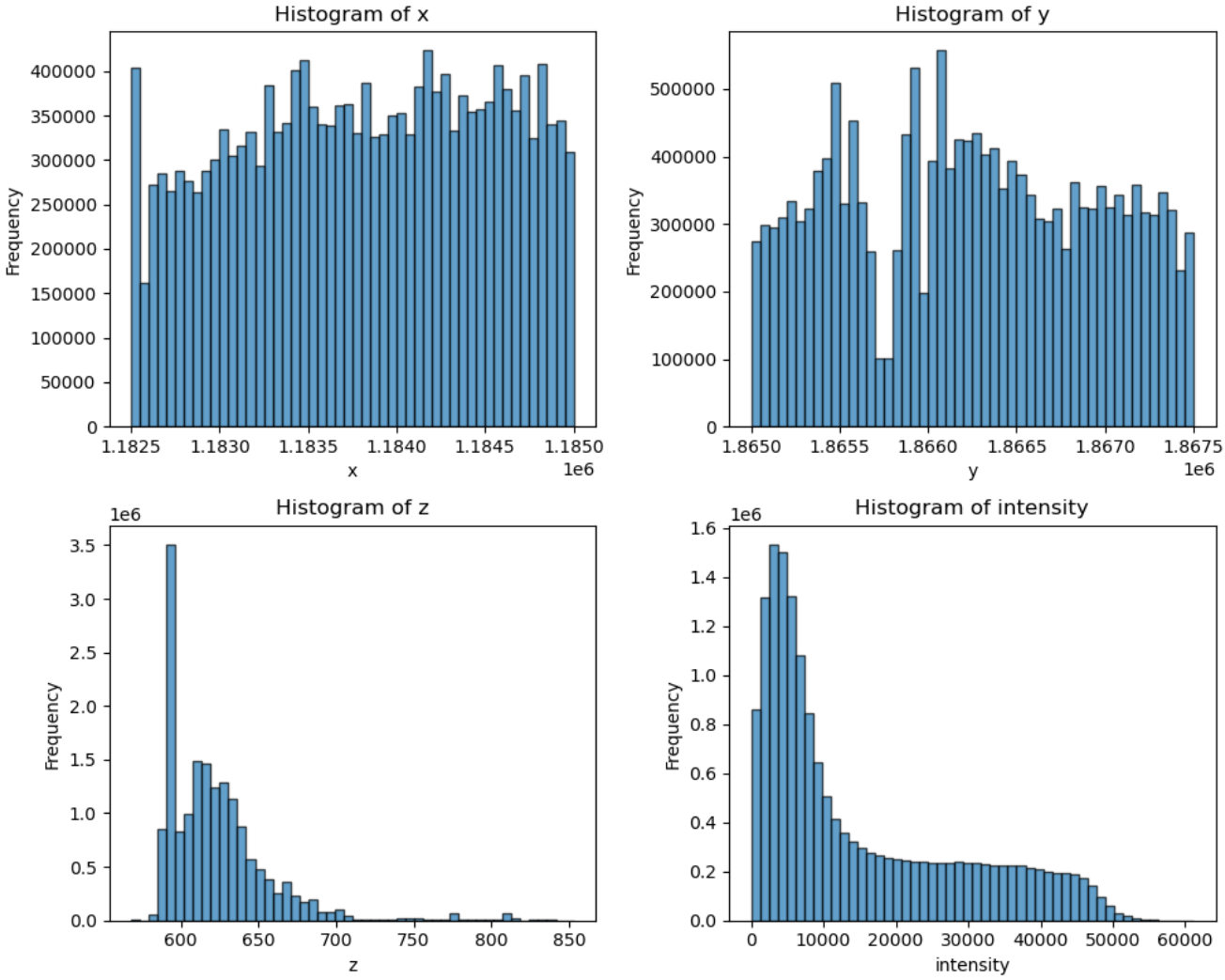
```
correlation_matrix = df.corr()
print("Correlation Matrix:")
print(correlation_matrix)
```

```
Correlation Matrix:
                   x         y         z  intensity
x           1.000000  0.012983 -0.172259  -0.046676
y           0.012983  1.000000  0.219634   0.076148
z          -0.172259  0.219634  1.000000  -0.083380
intensity -0.046676  0.076148 -0.083380   1.000000
```

The coefficient between x and y is 0.013, the correlation between x and z is -0.172, and the correlation between x and intensity is -0.047. It seems that there is a significant negative relationship between x and z since the coeefficient is -0.172.

3. Generate histograms for each column to visualize their distributions. Assess the histograms for any unusual patterns or anomalies.

```
columns = ["x", "y", "z", "intensity"]
plt.figure(figsize=(10, 8))

for i, column in enumerate(columns, 1):
    plt.subplot(2, 2, i)
    plt.hist(df[column], bins=50, edgecolor='k', alpha=0.7)
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```



As for x, it is roughly uniform. As for y, it seems that there is suddenly less data between 1.8655 x 10 ^6 and 1.8660 x 10 ^6. As for z, iy seems that there are lots of data around 550-600, but the number gradually decreases after around 620. As for intensity, at around 5000, the frequency reaches the peak. After that, the number gradually decreases with the increase of intensity. The frequency remians constant after intensity of 20000 and ends at 50000.

4. Reformat the 17-million point dataset into a 2,500 x 2,500 square grid representing the range of x and y. Aggregate the z and intensity values for points within each (x, y) pixel using a suitable aggregation function.

```
grid_size = 2500

x_min, x_max = df["x"].min(), df["x"].max()
y_min, y_max = df["y"].min(), df["y"].max()

x_bins = np.linspace(x_min, x_max, grid_size + 1)
y_bins = np.linspace(y_min, y_max, grid_size + 1)

df["x_bin"] = np.digitize(df["x"], x_bins) - 1
df["y_bin"] = np.digitize(df["y"], y_bins) - 1

df["x_bin"] = df["x_bin"].clip(0, grid_size - 1)
df["y_bin"] = df["y_bin"].clip(0, grid_size - 1)


grid = df.groupby(["x_bin", "y_bin"]).agg({
    "z": "mean",
    "intensity": "mean"
}).reset_index()

z_grid = grid.pivot(index="y_bin", columns="x_bin", values="z").values
intensity_grid = grid.pivot(index="y_bin", columns="x_bin", values="intensity").values
```

```
array([[591.        , 591.        ,          nan, ..., 614.9       ,
        613.5       , 618.3       ],
       [591.        , 591.        ,          nan, ..., 619.25      ,
        616.9       , 613.6       ],
       [591.        , 591.        ,          nan, ..., 611.22222222,
        615.14285714, 615.53846154],
       ...,
       [606.25      , 610.5       , 620.5       , ..., 606.16666667,
        603.5       , 599.375     ],
       [615.83333333, 620.        , 620.        , ..., 619.33333333,
        608.55555556, 603.66666667],
       [617.125     , 592.        , 613.        , ..., 625.2       ,
        624.28571429, 625.28571429]])
array([[39471.        , 40683.        ,           nan, ...,
        14896.7       , 15553.        , 10322.6       ],
       [39578.        , 39209.        ,           nan, ...,
         9172.        ,  8050.6       , 12942.4       ],
       [39895.        , 39823.        ,           nan, ...,
        10685.88888889,  7405.85714286,  9973.        ],
       ...,
       [23493.75      , 19545.66666667, 10382.5       , ...,
        13485.83333333, 11464.91666667, 14193.25      ],
       [16997.5       , 17302.33333333,  7471.33333333, ...,
        14696.33333333,  8891.11111111, 13431.33333333],
       [17634.5       , 33671.        , 15281.5       , ...,
        14788.2       , 11961.        , 13539.        ]])
```

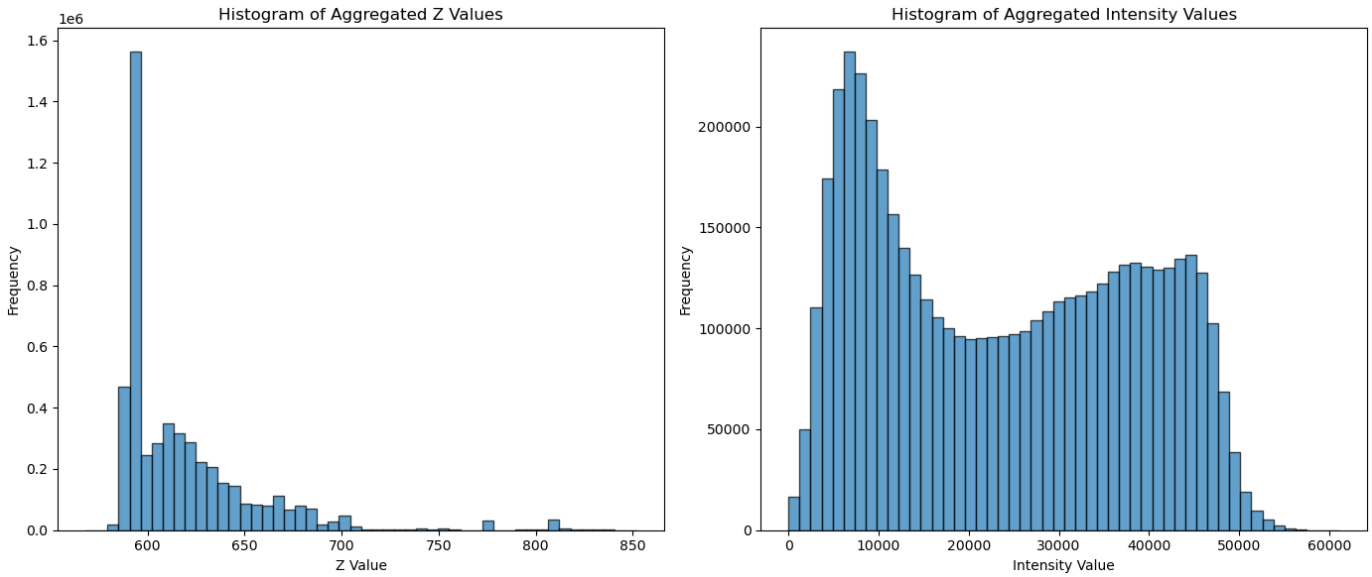
	x_bin	y_bin	z	intensity
0	0	0	591.000000	39471.000000
1	0	1	591.000000	39578.000000
2	0	2	591.000000	39895.000000

	x_bin	y_bin	z	intensity
3	0	4	591.000000	39280.000000
4	0	5	591.000000	39241.000000
...	...	...	...	...
5055956	2499	2495	588.500000	15774.250000
5055957	2499	2496	598.375000	7580.875000
5055958	2499	2497	599.375000	14193.250000
5055959	2499	2498	603.666667	13431.333333
5055960	2499	2499	625.285714	13539.000000

5055961 rows × 4 columns

5. Analyze the histograms of the aggregated z and intensity values for any suspicious patterns.

```
for _, row in grid.iterrows():
    x_idx = int(row['x_bin'])
    y_idx = int(row['y_bin'])
    z_grid[y_idx, x_idx] = row['z']
    intensity_grid[y_idx, x_idx] = row['intensity']

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
plt.hist(z_grid.ravel(), bins=50, edgecolor='k', alpha=0.7)
plt.title('Histogram of Aggregated Z Values')
plt.xlabel('Z Value')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.hist(intensity_grid.ravel(), bins=50, edgecolor='k', alpha=0.7)
plt.title('Histogram of Aggregated Intensity Values')
plt.xlabel('Intensity Value')
plt.ylabel('Frequency')

plt.tight_layout()
plt.show()
```

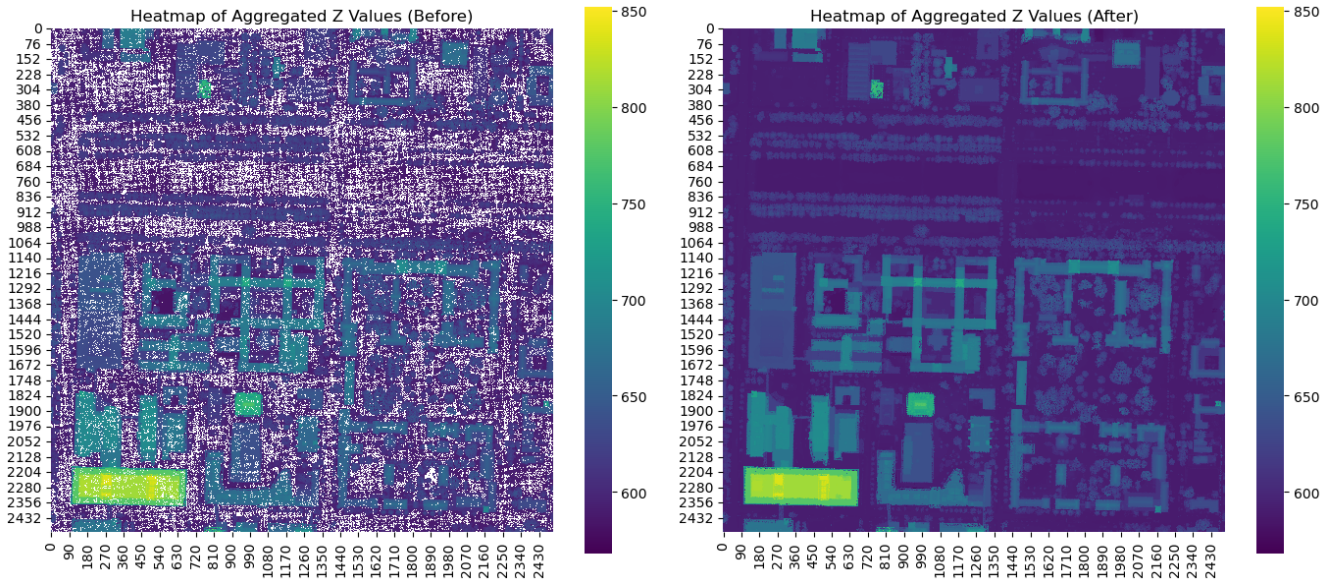


The aggregated z has a lots around 590. After than that, it gradually decreases. As for aggregated intensity, it rougly shows a U shape. It gradually increases at the peak at the intensity value of around 9000. And then it starts to decrease and increases slowly at around 20000

6. Fill the gaps in the x, y data grid (referred to as ”voids”), by interpolating missing values using nearby data points. Determine the method for selecting the replacement values.
7. Create heatmaps to visualize the aggregate z values before and after applying the gap-filling algorithm. Compare the results to assess the effectiveness of your gap-filling method.

```
from scipy.interpolate import griddata
import seaborn as sns

x_coords, y_coords = np.meshgrid(np.arange(grid_size), np.arange(grid_size))

z_grid_filled = griddata(
    (grid["x_bin"], grid["y_bin"]),
    grid["z"],
    (x_coords, y_coords),
    method='linear'
)

z_grid_filled = griddata(
    (grid["x_bin"], grid["y_bin"]),
    grid["z"],
    (x_coords, y_coords),
    method='nearest'
)

plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.heatmap(z_grid, cmap='viridis', cbar=True, square=True)
```

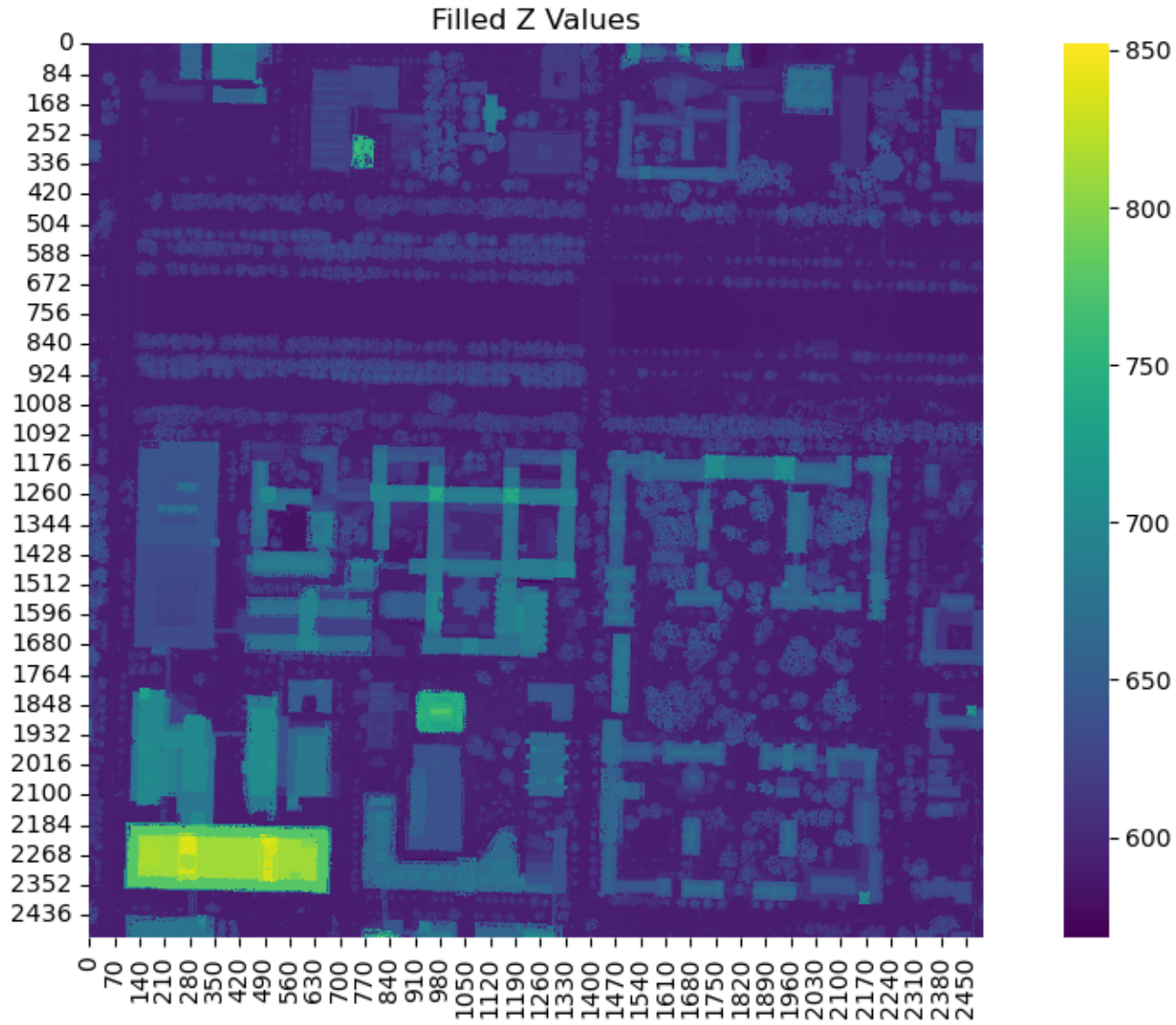
```
plt.title('Heatmap of Aggregated Z Values (Before)')

plt.subplot(1, 2, 2)
sns.heatmap(z_grid_filled, cmap='viridis', cbar=True, square=True)
plt.title('Heatmap of Aggregated Z Values (After)')

plt.tight_layout()
plt.show()
```

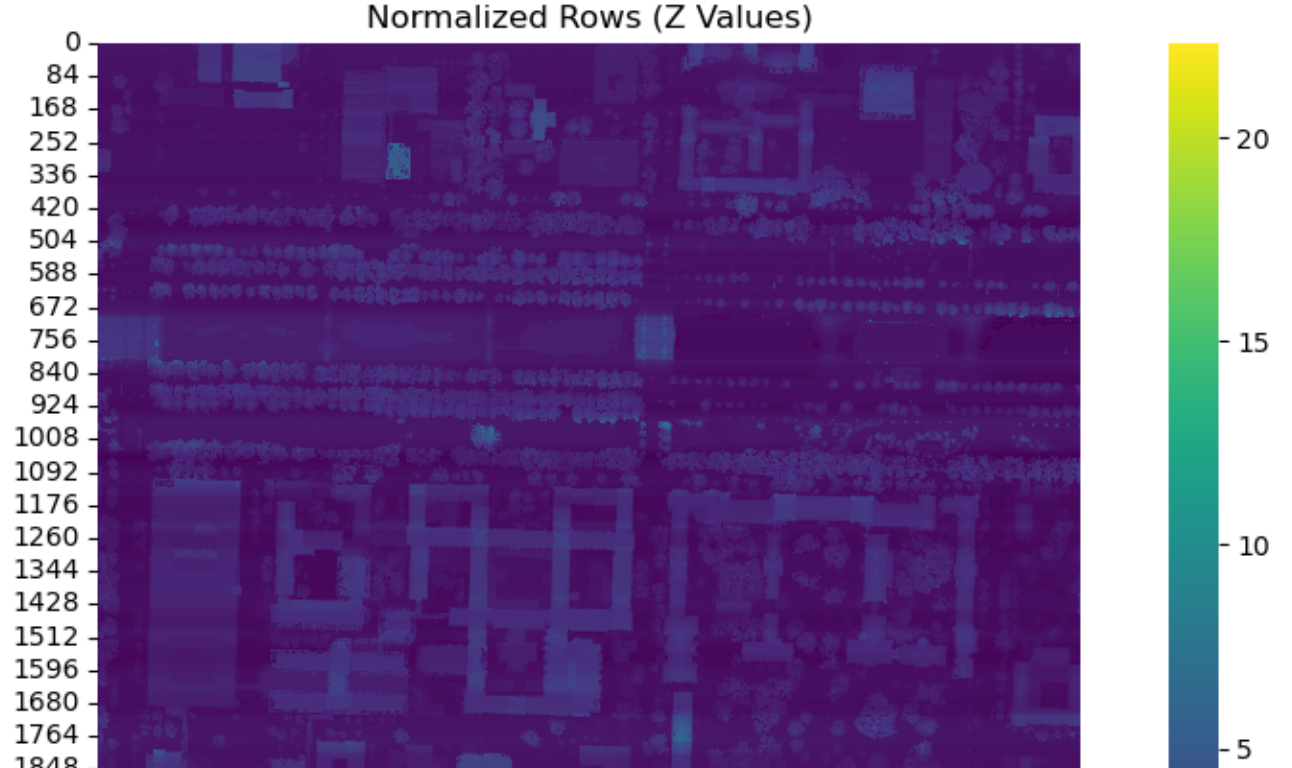


8. Under some circumstances, it is desired to have all of the rows or all the columns of a matrix have the same mean and variance. Write functions to transform the image so all the columns have the same variance, so all the rows have the same variance, and so that all the rows and columns have the same variance. Make a heatmap of one of the transformed images.

```
def normalize_columns(matrix):
    col_means = np.nanmean(matrix, axis=0)
    col_stds = np.nanstd(matrix, axis=0)
    return (matrix - col_means) / col_stds

def normalize_rows(matrix):
    row_means = np.nanmean(matrix, axis=1, keepdims=True)
    row_stds = np.nanstd(matrix, axis=1, keepdims=True)
    return (matrix - row_means) / row_stds


def normalize_rows_and_columns(matrix):
    matrix = normalize_rows(matrix)
    matrix = normalize_columns(matrix)
    return matrix


normalized_z_cols = normalize_columns(z_grid_filled)
normalized_z_rows = normalize_rows(z_grid_filled)
normalized_z_rows_cols = normalize_rows_and_columns(z_grid_filled)
```

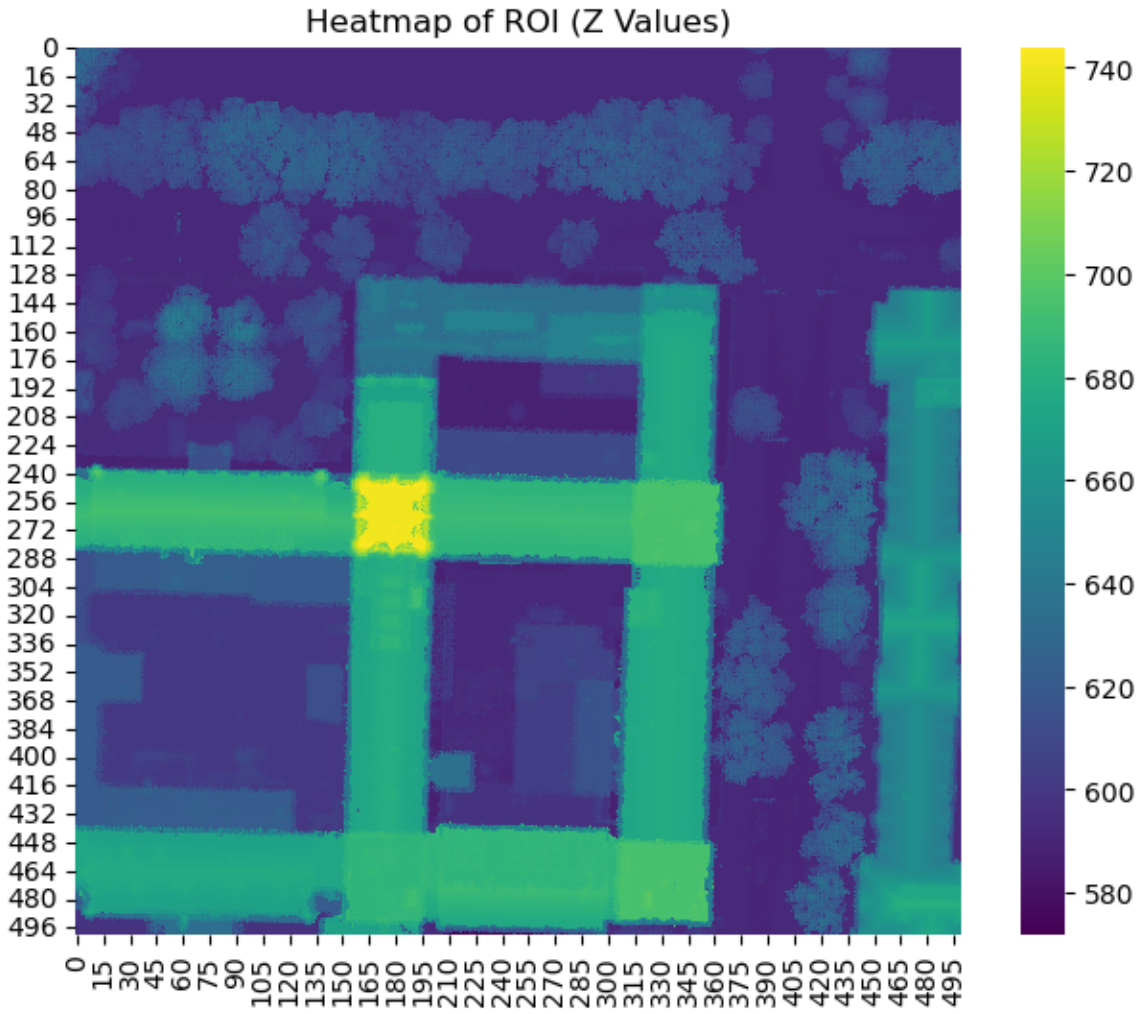
```
plt.figure(figsize=(18, 12))

plt.subplot(2, 2, 1)
sns.heatmap(z_grid_filled, cmap='viridis', cbar=True, square=True)
plt.title('Filled Z Values')

plt.subplot(2, 2, 2)
sns.heatmap(normalized_z_cols, cmap='viridis', cbar=True, square=True)
plt.title('Normalized Columns (Z Values)')

plt.subplot(2, 2, 3)
sns.heatmap(normalized_z_rows, cmap='viridis', cbar=True, square=True)
plt.title('Normalized Rows (Z Values)')

plt.subplot(2, 2, 4)
sns.heatmap(normalized_z_rows_cols, cmap='viridis', cbar=True, square=True)
plt.title('Normalized Rows and Columns (Z Values)')

plt.tight_layout()
plt.show()
```


Filled Z Values
0
84
168
252
336
420
504
588
672
756
840
924
1008
1092
1176
1260
1344
1428
1512
1596
1680
1764
1848
1932
2016
2100
2184
2268
2352
2436
0
70
140
210
280
350
420
490
560
630
700
770
840
910
980
1050
1120
1190
1260
1330
1400
1470
1540
1610
1680
1750
1820
1890
1960
2030
2100
2170
2240
2310
2380
2450
850
800
750
700
650
600


Normalized Rows (Z Values)
0
84
168
252
336
420
504
588
672
756
840
924
1008
1092
1176
1260
1344
1428
1512
1596
1680
1764
1848
20
15
10
5

9. Use numpy to extract a subset of the image representing a region of interest. Plot a heatmap for the ROI.

```
# Define the region of interest (ROI)
roi_start_x = 1000
roi_end_x = 1500
roi_start_y = 1000
roi_end_y = 1500


roi_z_grid = z_grid_filled[roi_start_y:roi_end_y, roi_start_x:roi_end_x]

plt.figure(figsize=(8, 6))
sns.heatmap(roi_z_grid, cmap='viridis', cbar=True, square=True)
plt.title('Heatmap of ROI (Z Values)')
plt.show()
```



10. Having familiarized yourself with the dataset, imagine you are developing a command-line API to process and visualize this data. Your task is to draft a specification for a command□line tool that accepts a dataset in the given format and produces image files. Consider what functionalities and options would be useful to

expose through your API. Draft your specifi□cation in the style of a manual page (manpage).

Note: You are not required to implement the actual image-generation or handle command□line arguments; merely outline how the proposed tool would function.

NAME: data_visualizer data.csv

SYNOPSIS: data_visualizer [OPTIONS]

DESCRIPTION: A detailed explanation of what the command does.

OPTIONS: -h –help Print online help.

-k –keepdate Keep the date stamp of output file same as input file.

-q –quiet Quiet mode. Suppress all warning and messages.

-V –version Prints version information.

-c –convmode convmode Sets conversion mode. Where convmode is one of: ASCII, 7bit, ISO, Mac with ASCII being the default. Simulates dos2unix under SunOS.

-o –oldfile file … Old file mode. Convert the file and write output to it. The program default to run in this mode. Wildcard names may be used.

-n –newfile infile outfile … New file mode. Convert the infile and write output to outfile. File names must be given in pairs and wildcard names should NOT be used or you WILL lose your files.

EXAMPLES: 1. Basic Usage: data_visualizer data.csv 2. Specify Output Directory: data_visualizer -o /path/to/output data.csv 3. Custom Grid Size and Interpolation Method: data_visualizer -g 3000 -m cubic data.csv 4. Normalize Rows and Generate Histograms: data_visualizer -n rows -z -i data.csv 5. Generate Heatmap for a Region of Interest: data_visualizer -r 1000,1500,1000,1500 -H data.csv

FUNCTIONALITY OUTLINE

Data Reading and Cleaning:

Reads the dataset from the input file. Converts columns to numeric types and handles non-numeric values by converting them to NaN. Drops rows with NaN values and resets the DataFrame index.

Grid Aggregation:

Aggregates the data into a 2D grid based on the specified grid size. Uses bins to digitize x and y coordinates and aggregates z and intensity values using the mean for each grid cell.
Interpolation of Missing Values:

Interpolates missing values in the grid using the specified interpolation method.

Normalization:

Normalizes the data along the specified dimension (rows, columns) to ensure uniform mean and variance.

Visualization:

Generates and saves heatmaps for the aggregated z values before and after interpolation and normalization. Generates and saves histograms for the aggregated z and intensity values before and after interpolation.

Region of Interest (ROI):

Allows the user to specify a region of interest within the grid. Extracts the subset of the grid corresponding to the ROI and generates a heatmap for this region.

AUTHOR: Lucy

SEE ALSO: matplotlib(1), numpy(1), scipy(1)