Lab 1 Report:

Data Preparation Techniques for Machine Learning

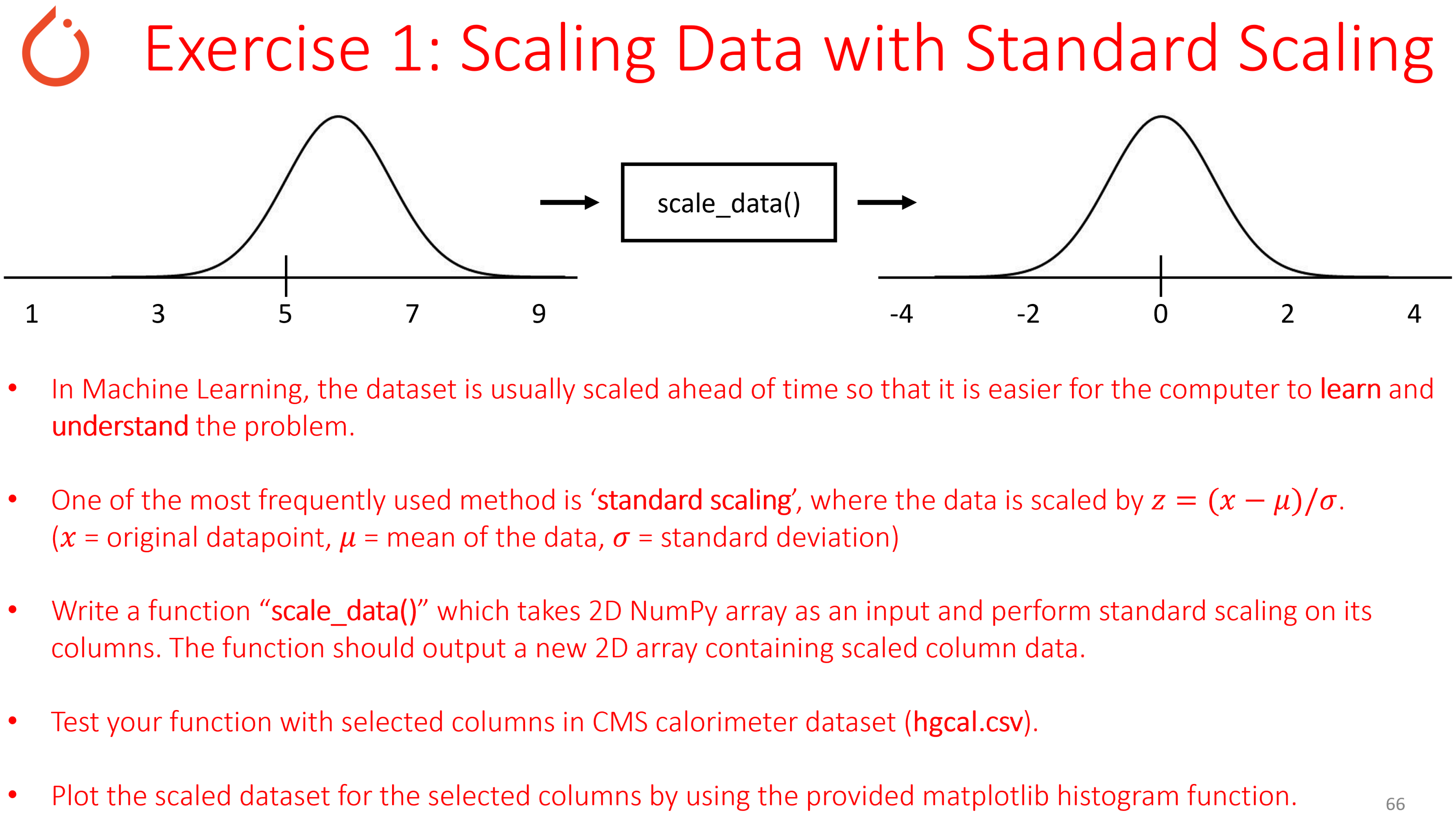
Name:

In [1]:
```
# Import necessary libraries

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

In [2]:
```
from IPython.display import Image # For displaying images in colab jupyter cell
```

In [3]:
```
Image('lab1_exercise1.PNG', width = 1000)
```

Out[3]:



In [4]:

```python
# Load the dataset (.csv) using pandas package

CMS_calori_dataset = pd.read_csv('hgcal.csv')

# .head directive on the panda dataframe displays the first n-rows

CMS_calori_dataset.head(n = 10)
```

Out[4]:

	Unnamed: 0	x	y	z	eta	phi	energy	trackId
0	0	179.50383	-23.632137	-7.878280	-0.0435	-0.130900	0.200126	462412
1	1	-143.63881	110.217940	-72.706795	-0.3915	2.487094	2.734594	493395
2	2	179.50383	-23.632120	-146.429610	-0.7395	-0.130900	0.423910	1
3	3	-172.67310	54.443620	-238.065340	-1.0875	2.836160	0.713950	493640
4	4	-180.88046	7.897389	-238.065340	-1.0875	3.097959	0.000000	495225
5	5	-180.88045	-7.897438	-238.065340	-1.0875	-3.097959	0.034491	495225
6	6	-152.69838	-97.279590	-265.020540	-1.1745	-2.574361	0.580138	460126
7	7	-23.63213	179.503810	-325.172060	-1.3485	1.701696	0.411487	465028
8	8	-152.69835	97.279594	89.977780	0.4785	2.574361	0.183141	1383
9	9	-176.76110	39.187016	107.930240	0.5655	2.923426	0.337551	4421

In [5]:

```
# Convert the panda dataframe into numpy 2D array

CMS_calori_dataset_np = CMS_calori_dataset.to_numpy()

# The converted numpy array has the dimension of 420 (rows) x 8 (columns)

print(CMS_calori_dataset_np.shape)
```

```
(420, 8)
```

In [6]:

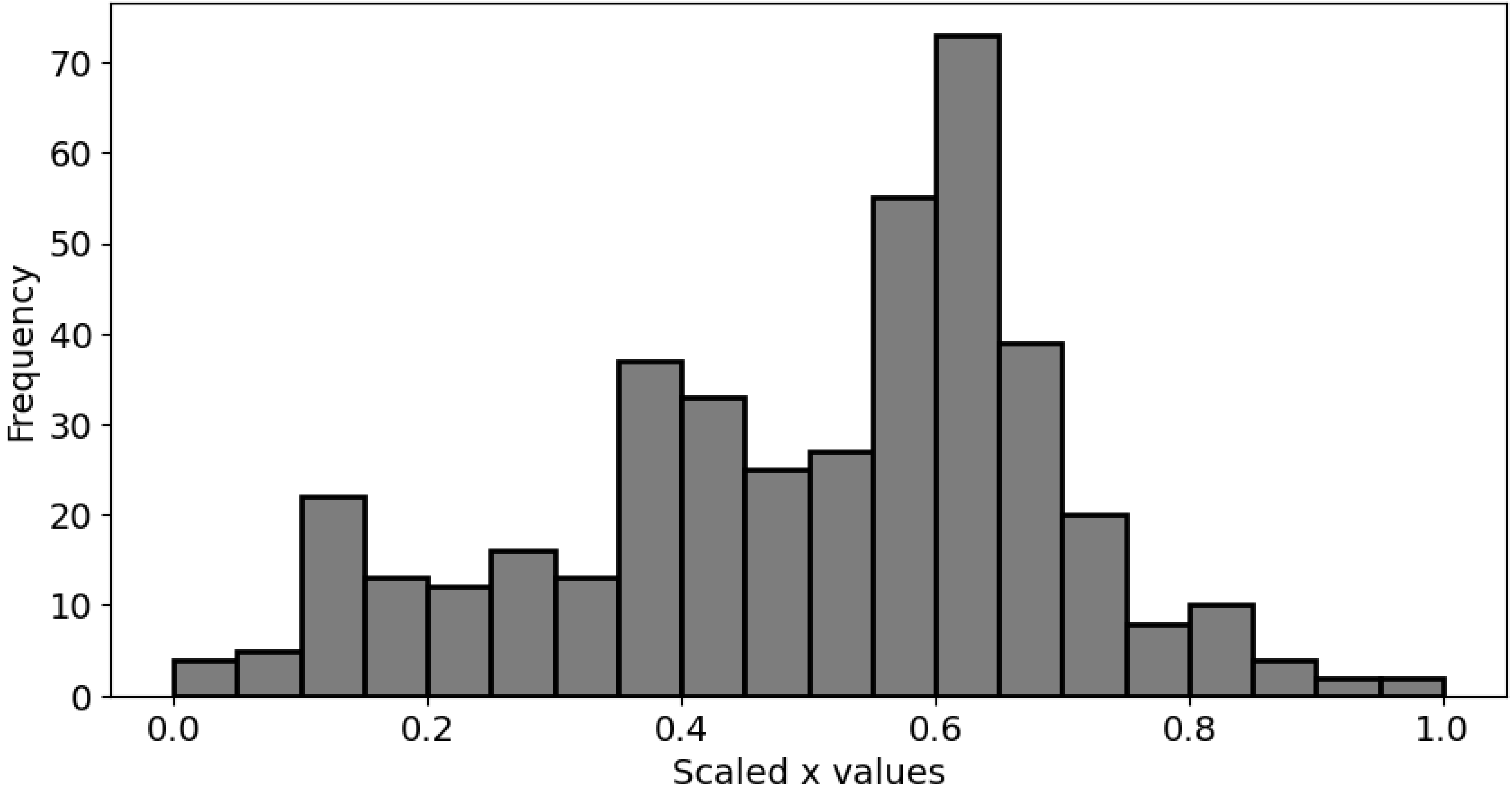
```
# Extract only x, y, z, eta, phi and energy columns from the dataset and stack them along column direction
# Name this new 2D array CMS_calori_dataset_np_sub.
# The array should have dimension 420 (rows) x 6 (columns)

# YOUR CODE HERE

# Extract the desired columns into a new DataFrame, then convert to NumPy
CMS_calori_dataset_np_sub = CMS_calori_dataset[['x', 'y', 'z', 'eta', 'phi', 'energy']].to_numpy()

# Verify the shape
print(CMS_calori_dataset_np_sub.shape)  # should print (420, 6)
```

```
(420, 6)
```

In [7]:
```
# Create the scaling function

def scale_data(arr):
    # Compute the minimum and maximum values for each column
    arr_min = arr.min(axis=0)
    arr_max = arr.max(axis=0)

    # Calculate the denominator, and replace zeros with 1 to avoid division by zero
    denom = arr_max - arr_min
    denom[denom == 0] = 1  # This ensures constant columns don't cause a division error

    # Apply the min-max scaling formula
    scaled_data = (arr - arr_min) / denom

    return scaled_data
```

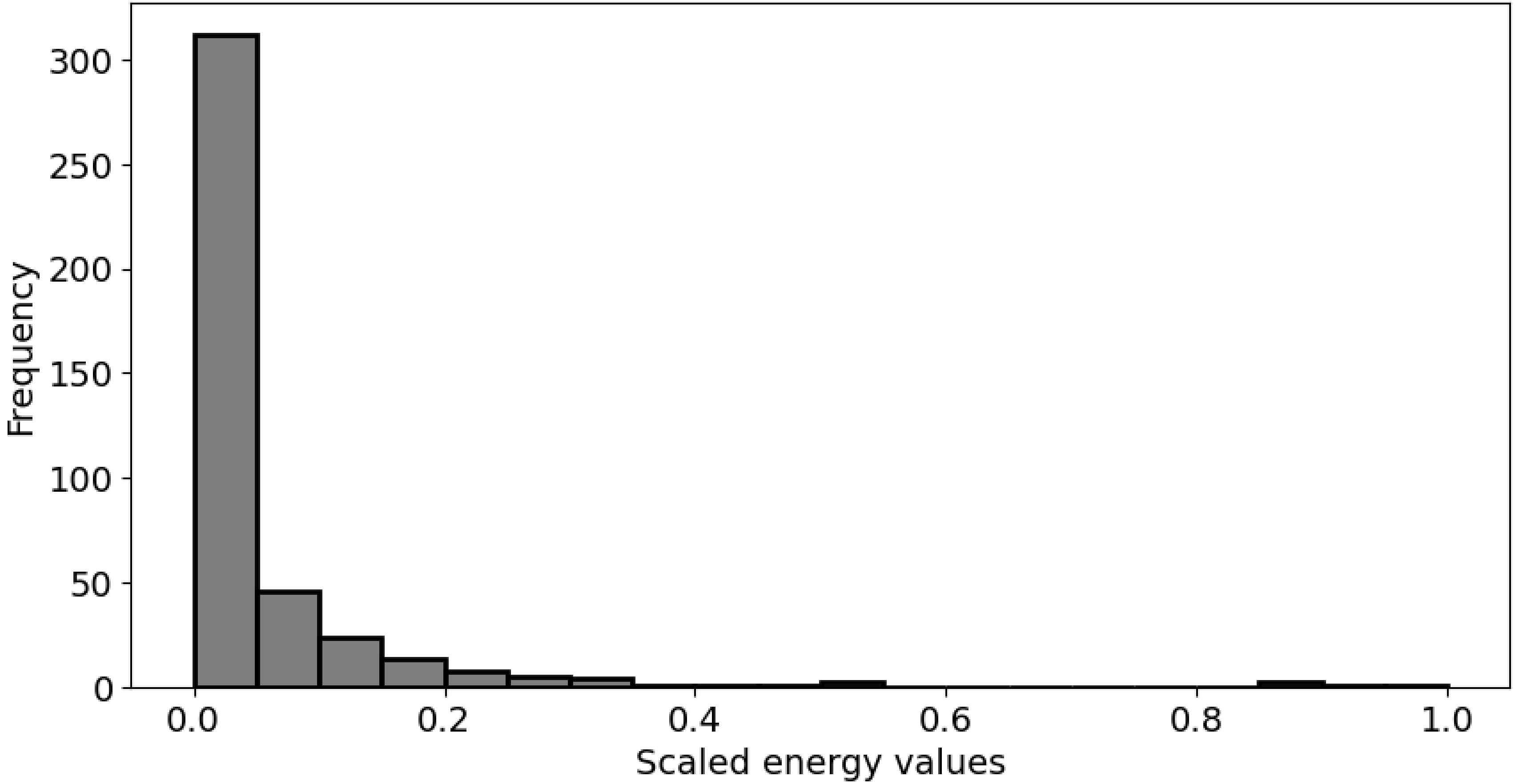
In [8]:
```
# Test the function with CMS_calori_dataset_np_sub

CMS_calori_dataset_np_sub_scaled = scale_data(CMS_calori_dataset_np_sub)
```

In [9]:
```
# Confirm the data is scaled for 'x' column

plt.figure(figsize = (10, 5))

plt.hist(CMS_calori_dataset_np_sub_scaled[:, 0], bins = 20, facecolor = 'grey', edgecolor = 'black', linewidth = 2)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

# YOUR CODE HERE
# Add proper x-label and y-label
plt.xlabel('Scaled x values', fontsize=14)
plt.ylabel('Frequency', fontsize=14)

plt.show()
```



In [10]:
```
# Confirm the data is scaled for 'energy' column

plt.figure(figsize = (10, 5))

plt.hist(CMS_calori_dataset_np_sub_scaled[:, 5], bins = 20, facecolor = 'grey', edgecolor = 'black', linewidth = 2)
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

# Add proper x-label and y-label

# YOUR CODE HERE
plt.xlabel('Scaled energy values', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
```

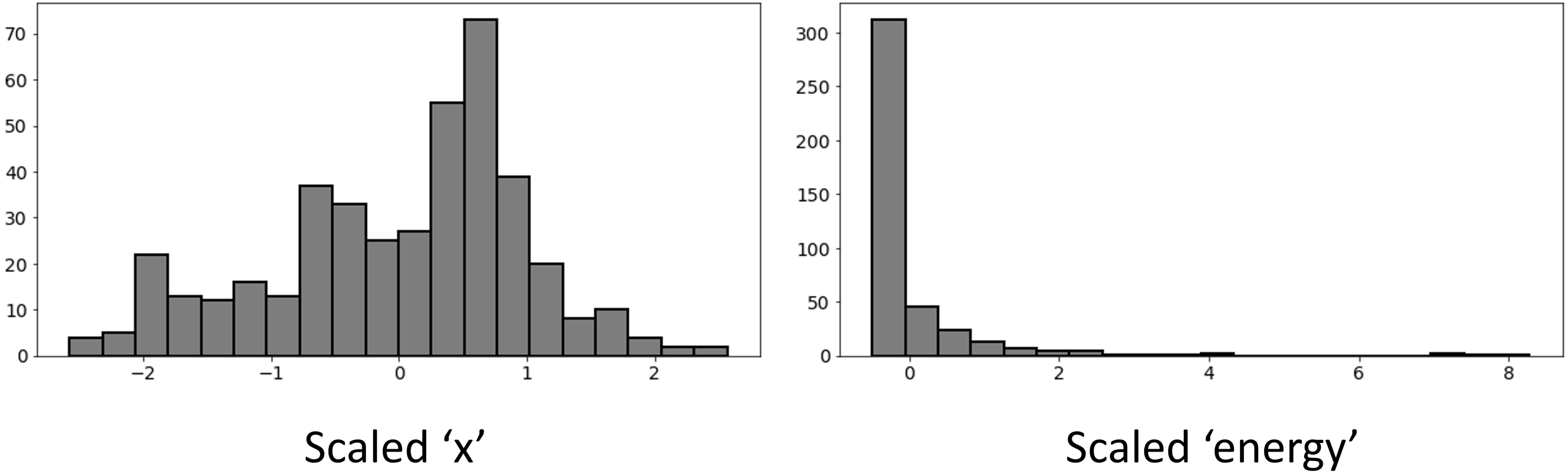
```
plt.show()
```



Expected histogram outputs - Feel free to style your plot differently

In [11]:
```
Image('lab1_e1_expected_outputs.PNG', width = 1000)
```

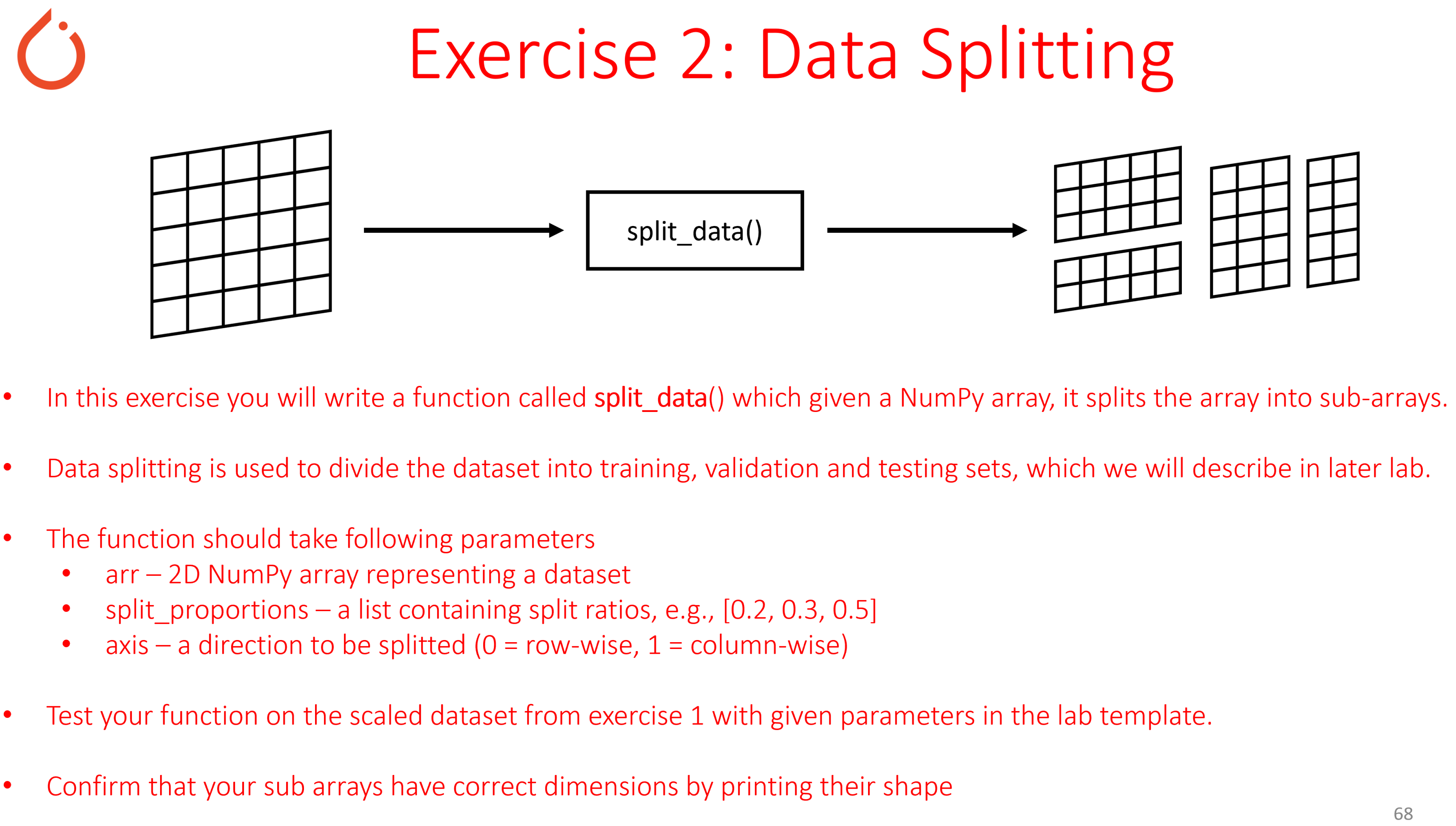
Out[11]:



In [12]:

```
Image('lab1_exercise2.PNG', width = 1000)
```

Out[12]:



In [13]:

```python
# Create the splitting function

def split_data(arr, split_proportions, axis):
    # Check if the sum of split proportions is equal to 1.0
    if not np.isclose(sum(split_proportions), 1.0):
        raise ValueError("Split proportions must sum to 1.0")

    # 'total' is the number of elements along the specified axis.
    # For example, if arr has shape (420, 6):
    # - For axis=0, total = 420 (number of rows)
    # - For axis=1, total = 6  (number of columns)
    total = arr.shape[axis]
```

```
    # Compute cumulative indices for splitting:
    # np.cumsum calculates the cumulative sum of the split proportions.
    # Multiplying by 'total' converts the proportions into index positions.
    cumulative = np.cumsum(split_proportions) * total

    # Round the cumulative indices to the nearest integer and convert them to integer type.
    cumulative = np.round(cumulative).astype(int)

    # Initialize an empty list to store the sub-arrays.
    split_data_list = []

    # Set the starting index to 0 for slicing.
    start_index = 0

    # Iterate over each cumulative (end) index.
    for end_index in cumulative:
        # Slice the array based on the specified axis.
        if axis == 0:
            # Row-wise splitting: select all columns for rows between start_index and end_index.
            sub_array = arr[start_index:end_index, :]
        elif axis == 1:
            # Column-wise splitting: select all rows for columns between start_index and end_index.
            sub_array = arr[:, start_index:end_index]
        else:
            raise ValueError("Axis must be 0 (rows) or 1 (columns)")

        # Append the sliced sub-array to the list.
        split_data_list.append(sub_array)

        # Update start_index for the next split.
        start_index = end_index

    # Return the list containing all the sub-arrays.
    return split_data_list
```

In [14]:
```
# Test your split function against scaled CMS Calorimieter dataset from exercise 1

sub_data_list_1 = split_data(arr = CMS_calori_dataset_np_sub_scaled,
                                    split_proportions = [0.6, 0.2, 0.2], axis = 0)
```

In [15]:
```
# Confirm that dataset has been split into correct shapes
# The correct dimensions should be (252, 6) (84, 6) (84, 6)
```

```
print(sub_data_list_1[0].shape, sub_data_list_1[1].shape, sub_data_list_1[2].shape)
```

```
(252, 6) (84, 6) (84, 6)
```

In [16]:
```
# Test your split function against scaled CMS Calorimieter dataset from exercise 1

sub_data_list_2 = split_data(arr = CMS_calori_dataset_np_sub_scaled,
                                                split_proportions = [0.5, 0.5], axis = 1)
```

In [17]:
```
# Confirm that dataset has been split into correct shapes
# The correct dimensions should be (420, 3) (420, 3)

print(sub_data_list_2[0].shape, sub_data_list_2[1].shape)
```

```
(420, 3) (420, 3)
```