Lab 1 Report:

Data Preparation Techniques for Machine Learning

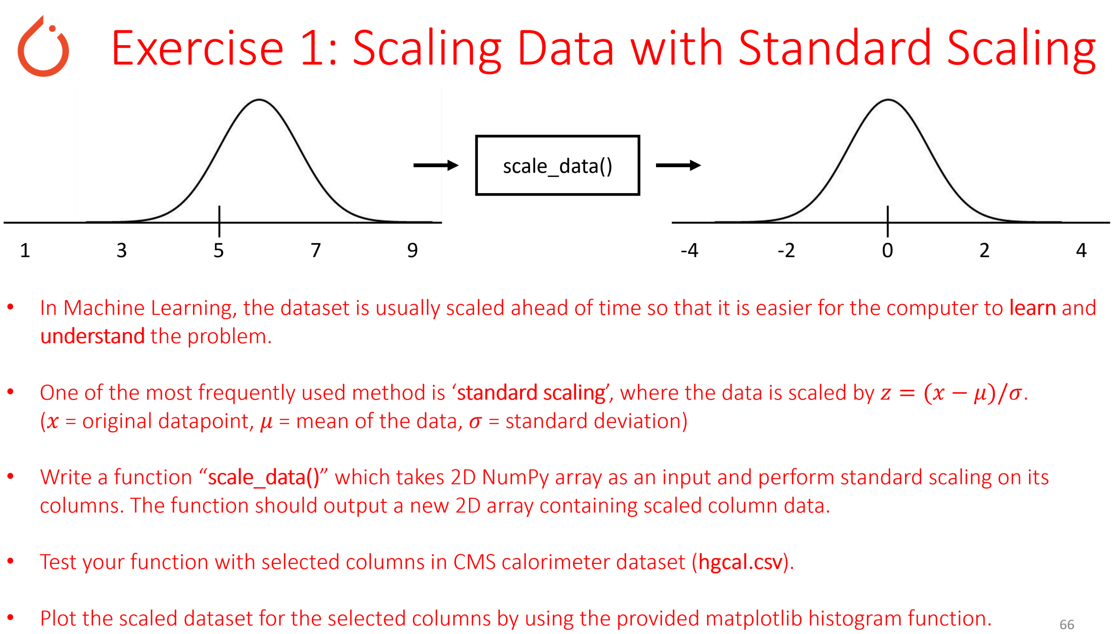
Name: Forest Tschirhart

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 c
```

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

Out[3]:



In [4]:
```
# 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()
#print(CMS_calori_dataset_np)

# 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 sta
# Name this new 2D array CMS_calori_dataset_np_sub.
# The array should have dimension 420 (rows) x 6 (columns)

CMS_calori_dataset_np_sub = CMS_calori_dataset_np[:,1:-1] # data extraction
print(CMS_calori_dataset_np_sub.shape)
```

```
(420, 6)
```

In [7]:
```
# Create the scaling function
# this does the assigned mathematical function elementwise on arr.
# However it expects a specific array shape and will break if given wrong sh
def scale_data(arr):
    scaled_data = []    # init empty list
    dim = arr.shape[1]  # get the number of columns
    for i in range(dim): # for each data type (x, y, z, etc.) apply respecti
        mean = np.mean(arr[:, i])
        std = np.std(arr[:, i])
        temp_arr = (arr[:, i] - mean) / std
        scaled_data.append(temp_arr)    # fill empty list
    scaled_data = np.array(scaled_data) # convert list to array
    return np.transpose(scaled_data) # transpose the array to get the correc
```

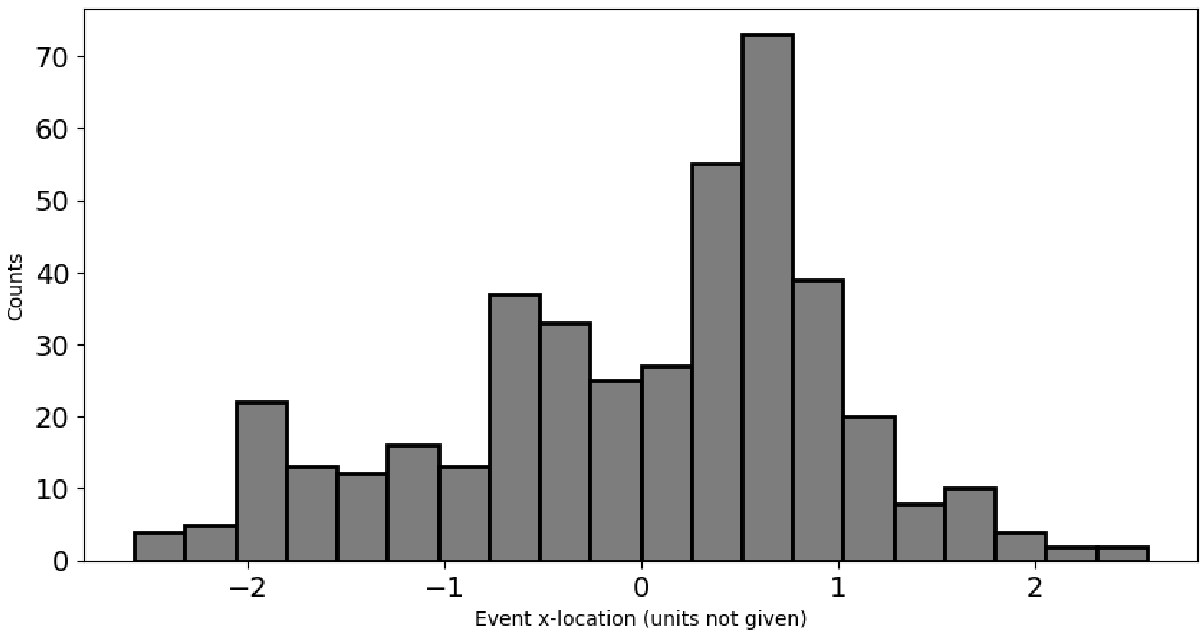
In [8]:
```
# Test the function with CMS_calori_dataset_np_sub
```

```
CMS_calori_dataset_np_sub_scaled = scale_data(CMS_calori_dataset_np_sub)
print(CMS_calori_dataset_np_sub_scaled[0])
```

```
[ 1.91214438 -0.51027049 -0.44193343 -0.47341363 -0.31488841 -0.38410307]
```

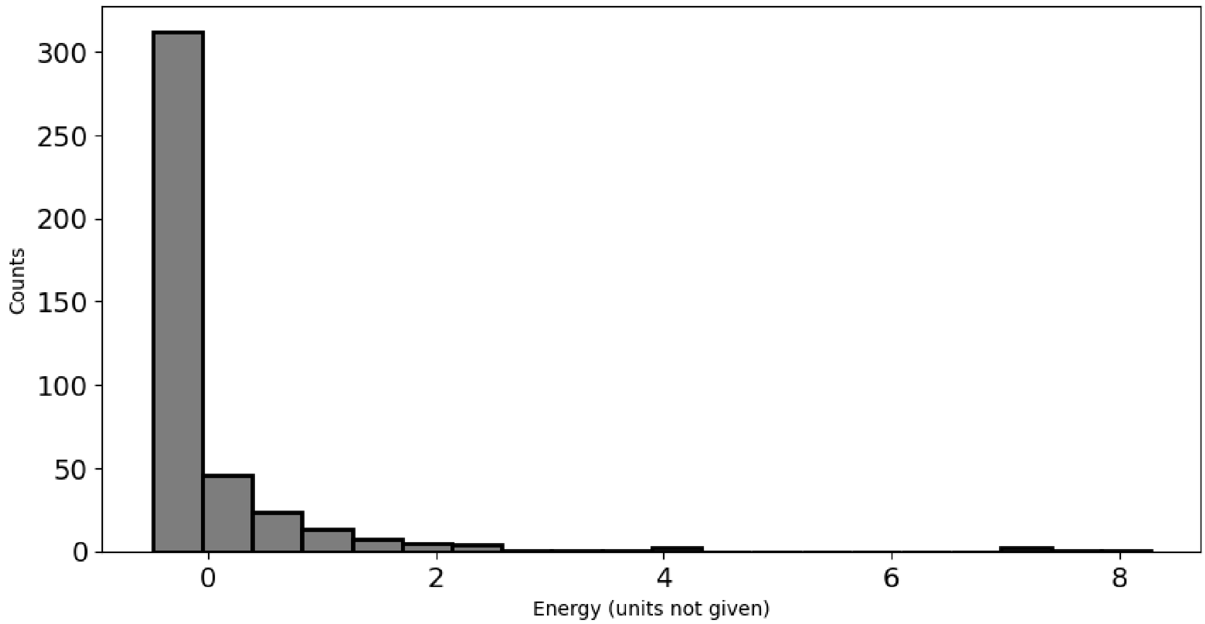
In [9]:

```
# 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 = 'gre
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

plt.xlabel('Event x-location (units not given)') # Units not given in assigr
plt.ylabel('Counts')

plt.show()
```



In [10]:

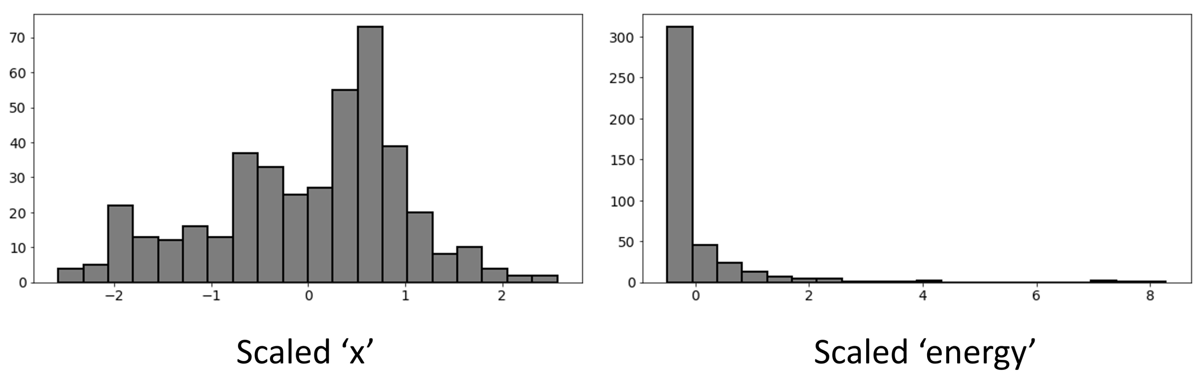
```
# Confirm the data is scaled for 'energy' column

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

plt.hist(CMS_calori_dataset_np_sub_scaled[:, 5], bins = 20, facecolor = 'gre
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

plt.xlabel('Energy (units not given)')
plt.ylabel('Counts')

plt.show()
```



Expected histogram outputs - Feel free to style your plot differently

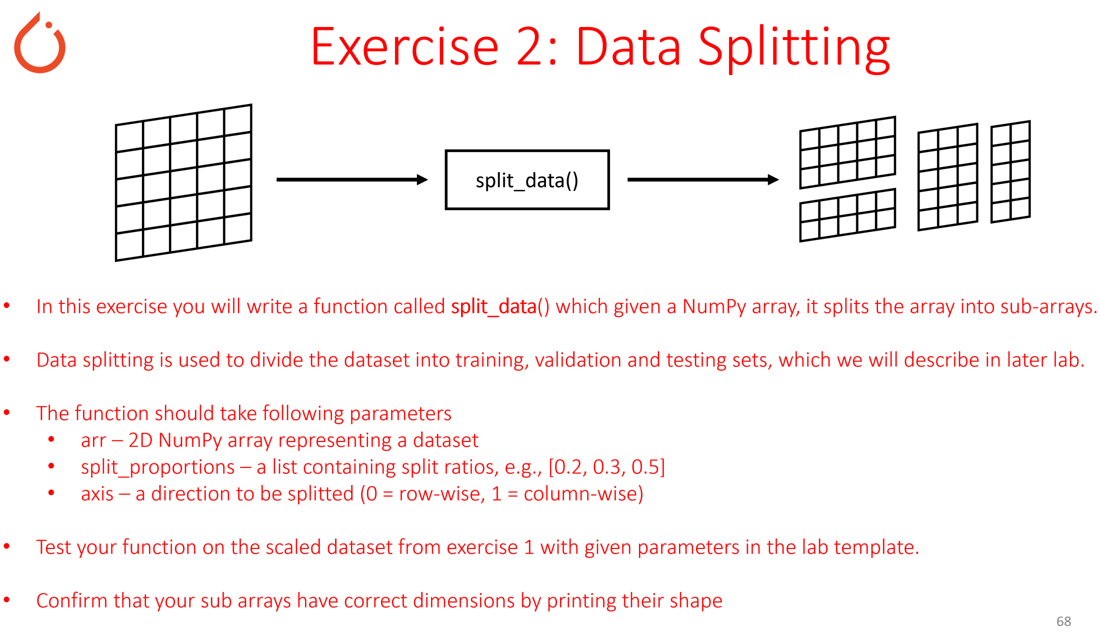
In [11]:
```
# Both histograms agree with my data manipulation

Image('lab1_e1_expected_outputs.png', width = 1000)
```

Out[11]:



In [12]:
```
Image('lab1_exercise2.png', width = 1000)
```

Out[12]:



In [13]:

```
# Create the splitting function
# Wrote this before I knew about np.split
def split_data(arr, split_proportions, axis):

    if not np.isclose(np.sum(split_proportions), 1):    # make sure split pr
        raise ValueError("Try again. The sum of split proportions must be ec

    # normalize proportions to array size. Convert to int for indexing
    slice_sizes = (np.array(split_proportions) * arr.shape[axis]).astype(int
    slice_sizes = slice_sizes[:-1] # last element not needed in loop

    split_data_list = []   # init empty list to be filled by for loop
    for i in slice_sizes:
        if axis == 0: # make sure slicing syntax is correct for desired axis
            temp_arr = arr[:i,:] # grab specified portion
            arr = arr[i:,:] # remove the grabbed portion from the original a
                            # that next iteration can also start from beginr
        else:
            temp_arr = arr[:,:i]
            arr = arr[:,i:]
        split_data_list.append(temp_arr)  # fill the list with the sliced da

    split_data_list.append(arr) # append the leftover portion to the split c

    return split_data_list


# This is a better way of doing the same thing but I kept the above functior
def split_data_numpy(arr, split_proportions, axis):

    # normalize proportions to array size. Convert to int for indexing
    if not np.isclose(np.sum(split_proportions), 1):
        raise ValueError("Try again. The sum of split proportions must be ec

    # Convert proportions to index numbers of the source array. eg. [0.2, 0.
    indices = (np.cumsum(np.array(split_proportions)) * arr.shape[axis]).ast
```

```python
    indices = indices[:-1] # last element not needed
    split_data_list = np.split(arr, indices, axis=axis) # nice and easy nump
    return split_data_list
```

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

sub_data_list_1 = split_data(arr = CMS_calori_dataset_np_sub_scaled,
                                    split_proportions = [0.6, 0.2, 0.2], axi
```

In [15]:
```python
# 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]
```

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

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

sub_data_list_2 = split_data(arr = CMS_calori_dataset_np_sub_scaled,
                                                split_proportions = [0.5, 0.
```

In [17]:
```python
# 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)
```

In [ ]:

In [ ]: