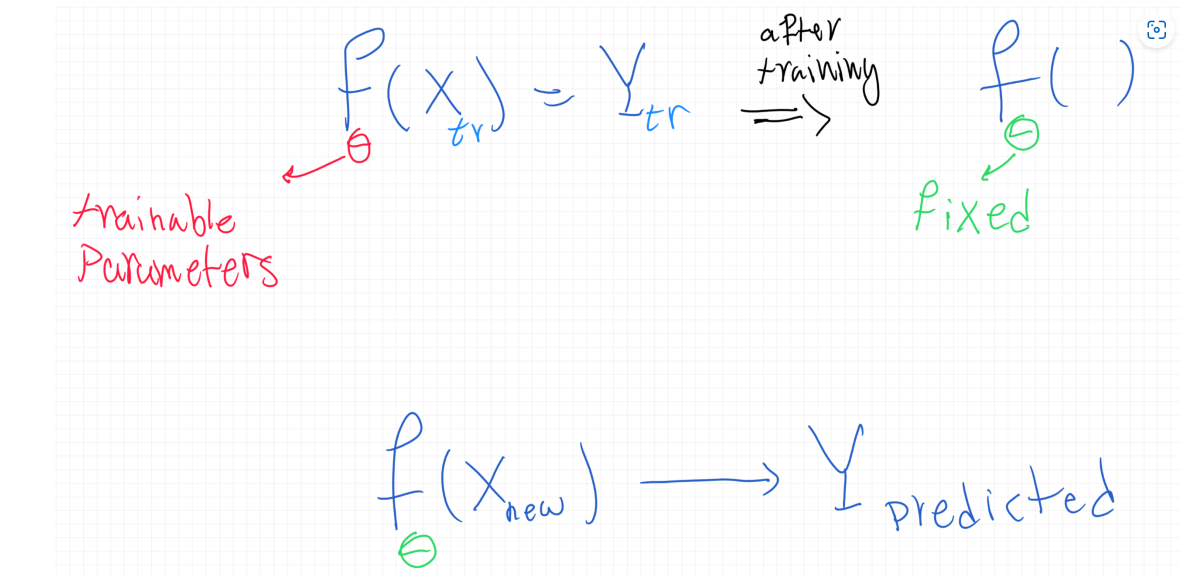
Course	Machine Learning Fundamentals (ML)
Instructor	Hossen Teimoorinia

May 13, 2023

==========

==========

==========



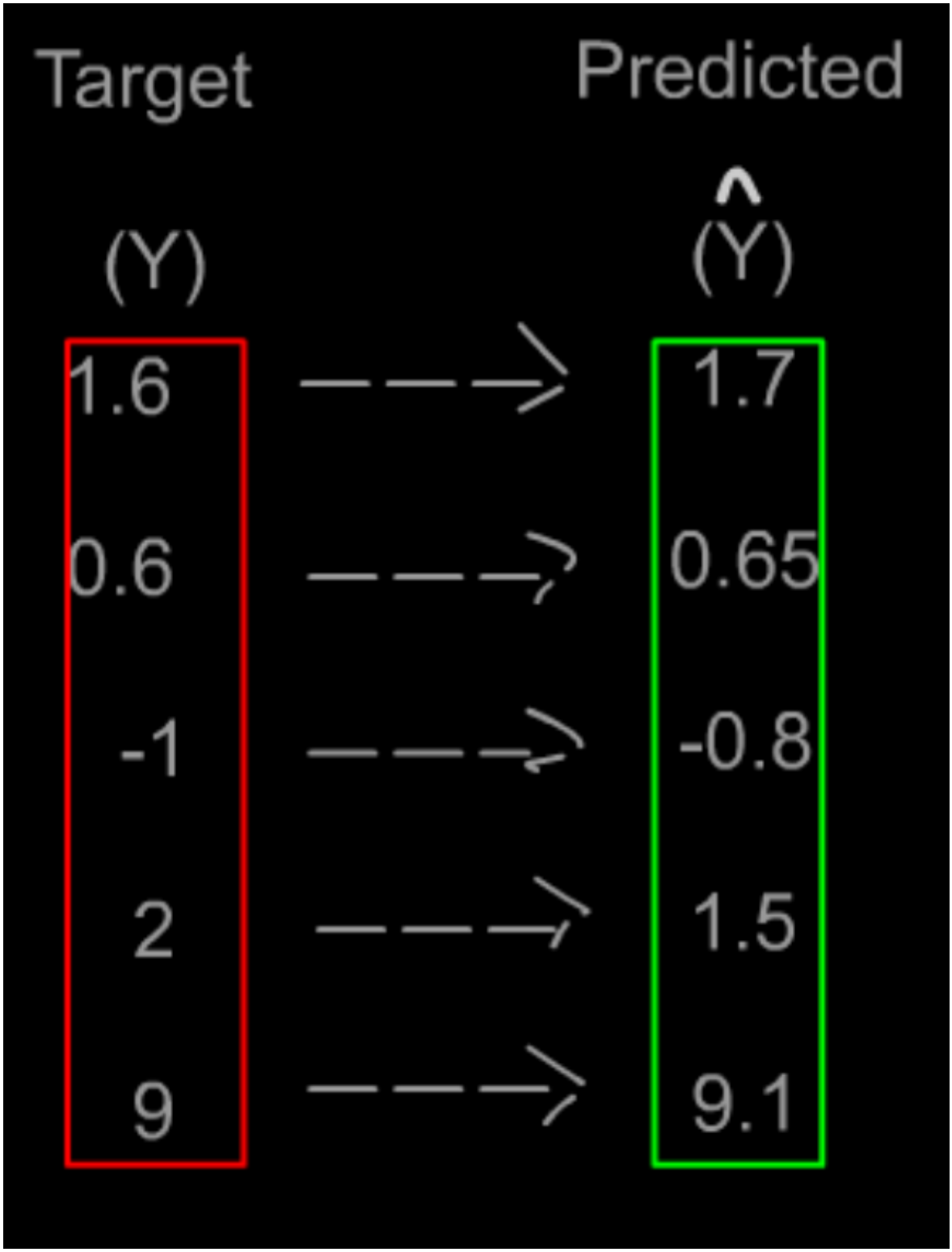
In [ ]:

A practical example in SKlrean

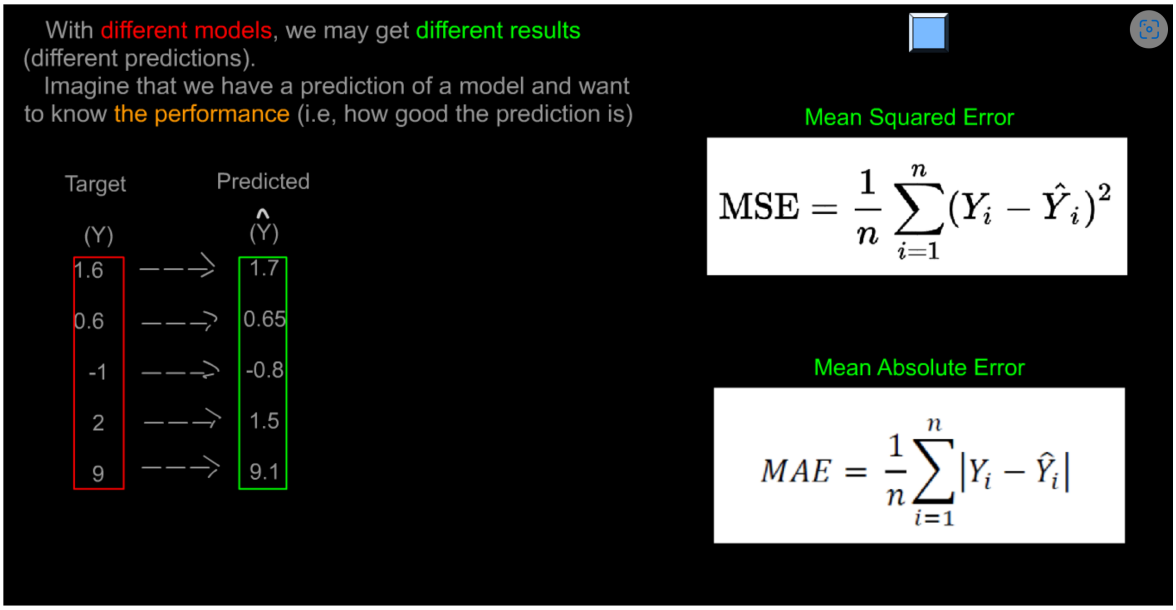
In [1]:

```
from sklearn import linear_model  # choose your model
reg = linear_model.Lasso(alpha=0.1)  # set up the model hyper-paramaters
reg.fit([[0, 0], [1, 1]], [0, 1])   # train
reg.predict([[1, 1]])  # predict
```

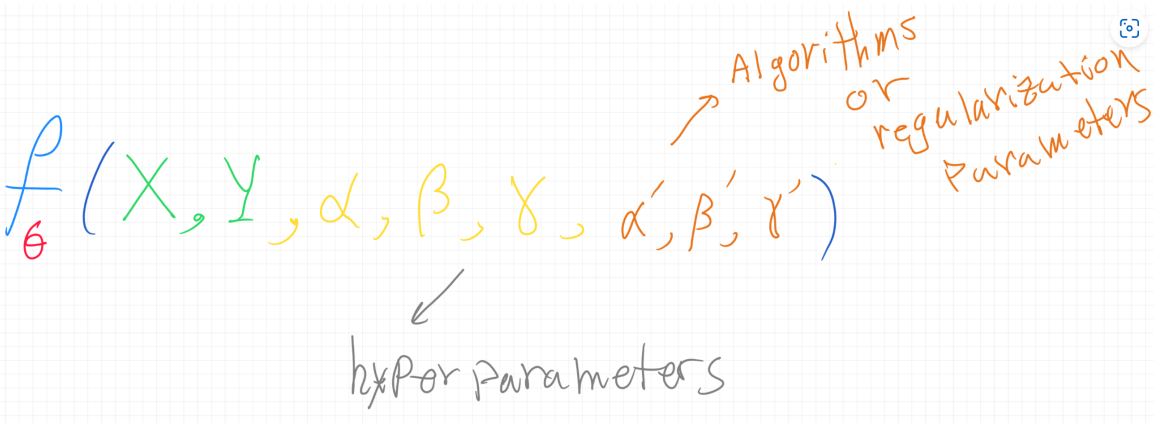
Out[1]: array([0.8])



Error / loss functions



A typical model (regression, classification or clustring)



Classification, regression and clustring models: What are practical steps in predictions (what to do)?

1- Define the problem: Clearly state the problem you want to solve, the inputs, and the desired outputs. Identify the type of machine learning task: classification, regression, clustering, etc.

2- Collect and preprocess data: Gather a dataset that is representative of the problem domain. Preprocess the data by cleaning, handling missing values, converting categorical data to numerical data, and normalizing or scaling the features if necessary.

3- Split the data: Divide the dataset into training, validation (optional), and testing subsets. A common split ratio is 70% for training, 30% for validation.

4- Select a model: Choose a suitable machine learning algorithm based on the problem and data. Examples include linear regression, logistic regression, support vector machines, decision trees, random forests, and neural networks.

5- Train the model: Feed the training data to the algorithm, allowing it to learn the underlying patterns in the data. This usually involves minimizing a loss function through an optimization process, such as gradient descent or another optimization algorithm.

6- Tune hyperparameters: Optimize the hyperparameters of the model using the validation set. This can involve techniques like grid search, random search, or Bayesian optimization.

7- Evaluate the model: Assess the performance of the trained model on the test set using appropriate evaluation metrics (e.g., accuracy, precision, recall, F1 score, mean squared error, etc.). This step helps you understand how well the model generalizes to new, unseen data.

8- Iterate and improve: If the model's performance is not satisfactory, you may need to iterate by revisiting steps 2-7. This can involve collecting more data, trying different algorithms, or further tuning hyperparameters.

9- Deploy the model: Once you're satisfied with the model's performance, deploy it to a production environment, where it can be used to make predictions on new data.

In order to proceed, it is crucial to emphasize the significance of data preparation and normalization as the second step.

1- Cleaning the data can be done by Pandas or Numpy

2 - Normalizatin can by done by SKlearn

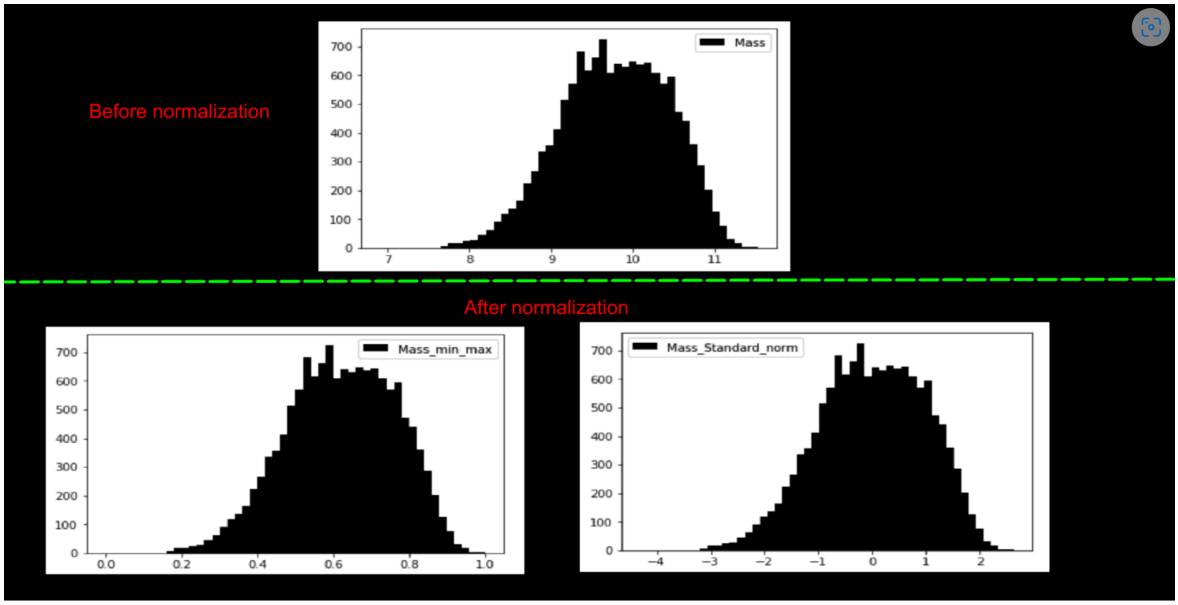
Rescaling (min-max normalization)

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Standardization (Z-score Normalization)

$$x' = \frac{x - \bar{x}}{\sigma}$$

65.78
71.52
69.40
68.22
67.79
68.70
69.80
70.01
67.90
66.78
66.49
67.62
68.30
67.12
68.28
71.09
66.46
68.65
71.23
67.13
67.83
68.88
63.48
68.42
67.63
67.21



A real and complex situation

tar_1	tar_2	tar_3	Tar_4	tar_5	tar_6	tar_7	tar_8	tar_9	tar_10	tar_11	tar_12	z	mass	agn_sn	cf_g	cf_r	Ha	Hb	OII_1	OII_2	OIII	NII	SII_1	SII_2	err_Ha	err_Hb	err_OII_1	err_OII_2	err_OIII	err_NII
nan	8.92791	8.54151	8.66548	8.66583	nan	nan	8.49927	8.44851	nan	8.7153	8.60276	0.087765	10.2193	5.69672	0.141788	0.173045	173.534	60.9436	84.3486	66.8856	33.756	51.5031	32.7227	30.5258	1.89123	1.71128	3.23533	3.1359	1.69458	1.37601
nan	8.79355	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	0.012842	9.06587	3.62486	0.049381	0.046234	45.7435	17.6716	0	0	10.2213	17.2484	13.5688	10.011	1.65269	1.90977	-0.73152	-0.73152	1.80062	1.37111
nan	9.06602	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	0.045416	10.2279	4.504	0.109658	0.125176	766.757	269.407	288.01	156.784	60.3379	259.768	130.521	85.9468	4.16357	2.86528	5.25365	5.05405	2.50198	2.80719
8.86975	8.88465	8.51746	8.6418	8.67329	8.96472	8.97729	8.48092	8.45596	8.81754	8.69972	8.56548	0.072148	9.84917	12.8796	0.199379	0.19366	514.525	180.658	268.702	249.1	84.2799	139.355	103.502	73.061	3.21591	2.55166	4.74407	4.65192	2.26811	1.82378
nan	8.85254	nan	8.7363	8.64105	nan	nan	8.54667	8.42411	nan	nan	8.71032	0.093481	10.3845	6.51815	0.344223	0.366067	286.553	100.622	132.1	78.6799	91.7679	107.707	67.0255	34.8271	4.67031	4.00723	7.6344	7.74191	4.43649	3.59679
9.10778	9.00766	8.84806	nan	8.74502	nan	nan	8.58946	8.53044	nan	8.90719	8.82278	0.038591	10.4623	7.59886	0.099568	0.11775	1011.95	355.587	494.637	464.84	127.852	470.79	197.252	124.14	4.53658	2.994	6.77311	6.97232	2.92078	3.46677
9.11417	8.94557	8.67884	8.78215	8.69734	nan	nan	8.56947	8.48038	nan	8.70028	8.76831	0.091271	10.4154	9.06507	0.316811	0.332568	284.017	99.6942	120.66	75.0398	55.603	119.6	69.9102	47.5061	2.99927	2.60283	4.35894	4.23139	2.75373	2.40426
nan	8.98596	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	0.126874	10.48	3.15379	0.168314	0.196137	240.176	84.368	70.6947	100.385	26.3161	78.7734	46.5706	28.376	3.04635	1.91218	2.83311	2.91534	1.92446	1.88157
9.08641	9.10769	8.79797	8.75374	8.77658	9.18361	9.12241	8.55619	8.56479	9.01591	8.87883	8.73402	0.081958	10.7657	6.56461	0.14832	0.155367	542.953	190.665	112.974	143.836	40.1321	213.997	87.8345	69.0888	3.14163	2.18222	3.4923	3.73229	1.88348	2.07972
nan	8.96692	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	0.129058	10.6135	4.88436	0.150231	0.151887	363.056	127.613	162.806	173.282	54.729	103.863	61.425	50.2906	1.80248	1.4162	2.3493	2.28025	1.47694	1.47655
nan	9.0475	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	0.157144	10.6589	3.89617	0.299751	0.306422	554.167	194.672	80.2988	228.585	65.2525	272.757	108.352	81.026	5.21758	3.42868	5.3125	5.76571	3.75096	4.07658
8.86924	8.86576	8.52105	8.62617	8.60989	8.92449	8.94912	8.46827	8.39429	8.82391	8.62553	8.54113	0.070113	10.0804	28.9003	0.224471	0.256401	1176.25	413.061	520.896	593.058	351.31	299.127	210.699	158.976	6.36969	3.51936	5.54937	5.68497	3.56658	2.83088
nan	8.98618	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	0.079637	10.487	3.62359	0.103238	0.118402	184.64	64.8643	69.4147	71.6608	26.1079	64.2464	35.1185	26.853	1.82351	1.61687	3.46735	3.49821	1.55487	1.4336
nan	8.492	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	0.078015	9.52978	3.98234	0.127221	0.145908	59.3078	20.8144	20.2926	47.6062	14.6466	10.0435	0	11.9582	1.58934	1.61883	2.88736	3.19544	1.61375	1.07968
nan	8.21889	7.94899	8.3895	8.4063	nan	nan	8.25603	8.22264	nan	8.31843	8.25543	0.021417	8.64675	8.00608	0.126485	0.12795	216.252	75.9098	0	0	142.137	19.095	38.841	27.7884	1.8307	1.85211	-0.67489	-0.67489	2.16264	0.90341
nan	8.13945	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	0.031546	8.92487	4.11733	0.095834	0.096232	130.169	45.7004	87.7629	111.674	65.7727	10.8016	22.4974	17.9617	1.495	1.66275	5.4655	5.4831	1.77499	0.895942
9.14462	9.12902	8.96768	8.84304	8.74273	nan	nan	8.58693	8.52799	nan	8.9224	8.81569	0.084896	10.7221	11.7649	0.291286	0.306874	2234.24	785.203	390.894	848.815	286.244	1026.4	294.215	236.006	7.5536	4.11465	6.52712	7.05075	3.68718	4.96791
nan	9.00966	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	0.108666	10.3534	3.83783	0.298071	0.287899	372.506	130.849	139.506	91.9403	34.8702	122.958	68.6289	43.9462	3.51686	2.2579	3.84667	3.8109	2.15259	2.29757
nan	8.93865	8.58861	8.68324	8.67136	nan	nan	8.51231	8.45403	nan	8.74536	8.63067	0.080731	10.1177	5.33296	0.216371	0.241746	228.693	80.3291	111.78	80.964	44.7404	72.4298	43.3655	37.8209	2.37052	2.13816	3.91634	3.89584	2.05076	1.71955
8.8382	8.79905	8.50839	8.61956	8.58922	8.86256	8.90155	8.46281	8.37503	8.79052	8.63089	8.53093	0.061841	9.73219	15.6189	0.212677	0.235266	629.751	221.209	272.559	390.421	225.25	155.85	122.859	74.0243	3.49924	2.70181	5.04251	5.47519	2.64321	1.92362
8.88956	8.94512	8.58523	8.65736	8.66615	8.96324	8.97649	8.49309	8.44882	8.82825	8.71439	8.58995	0.146101	10.2424	10.9611	0.322701	0.351625	479.47	168.384	194.802	274.527	90.1367	137.988	91.2849	60.7011	3.62478	2.56131	3.57899	3.78333	2.37503	2.008
8.83148	8.90623	8.62703	8.66548	8.62156	8.80144	8.84571	8.49927	8.40535	8.71595	8.75246	8.60275	0.150396	10.4168	12.1839	0.274992	0.29662	781.276	274.602	571.272	443.078	241.983	231.869	142.392	91.4557	3.39429	2.36215	3.52129	3.47664	2.73225	2.49207
nan	8.44024	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	0.032639	8.88428	4.79166	0.08745	0.087579	75.5126	26.499	37.0541	69.4248	42.8348	12.022	19.3052	11.3269	1.79843	1.83638	5.52409	5.80434	2.07602	1.1395
nan	8.32615	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	0.048634	9.01832	4.58876	0.222087	0.248405	154.8	54.3424	100.187	106.642	79.4829	18.4067	35.9441	24.0493	2.38403	1.99145	4.85189	5.07429	2.42899	1.45749
nan	9.04795	8.73689	8.71319	8.73168	nan	nan	8.53263	8.51622	nan	8.89812	8.67668	0.072371	10.3941	5.38493	0.146744	0.16148	715.806	251.55	324.753	203.619	79.4068	250.87	117.213	88.0686	3.75623	2.44043	4.14401	4.06386	2.34467	2.5451

In [2]:
```
import pandas as pd # to work with CSV files
# Read file 'Expert.csv' as data_exp
data_exp= pd.read_csv('Expert.csv')
print('the file loaded')
```

```
the file loaded
```

In [3]:
```
data_exp # See the table
```

Out[3]:

	tar_1	tar_2	tar_3	tar_4	tar_5	tar_6	tar_7	tar_8	tar_9	tar_1
0	NaN	8.92791	8.54151	8.66548	8.66583	NaN	NaN	8.49927	8.44851	Nal
1	NaN	8.79355	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
2	NaN	9.06602	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
3	8.86975	8.88465	8.51746	8.64180	8.67329	8.96472	8.97729	8.48092	8.45596	8.8175
4	NaN	8.85254	NaN	8.73630	8.64105	NaN	NaN	8.54667	8.42411	Nal
...	...	...	...	...	...	...	...	...	...	.
158883	8.94060	8.96944	8.60891	8.66408	8.72856	9.06701	9.05309	8.49821	8.51291	8.9068
158884	NaN	9.08881	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
158885	NaN	8.91223	8.60198	8.72192	8.69056	NaN	NaN	8.53811	8.47344	Nal
158886	NaN	9.07084	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Nal
158887	8.92745	8.95868	8.68229	8.68525	8.66979	8.98096	8.98990	8.51373	8.45245	8.8362

158888 rows × 33 columns

In [4]:
```
data_exp.info()  # get information and check NaNs
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 158888 entries, 0 to 158887
Data columns (total 33 columns):
 #   Column     Non-Null Count   Dtype  
---  ------     --------------   -----  
 0   tar_1      75852 non-null   float64
 1   tar_2      158852 non-null  float64
 2   tar_3      92991 non-null   float64
 3   tar_4      94951 non-null   float64
 4   tar_5      97650 non-null   float64
 5   tar_6      59447 non-null   float64
 6   tar_7      60146 non-null   float64
 7   tar_8      97634 non-null   float64
 8   tar_9      96436 non-null   float64
 9   tar_10     60306 non-null   float64
 10  tar_11     95643 non-null   float64
 11  tar_12     97047 non-null   float64
 12  z          158888 non-null  float64
 13  mass       158852 non-null  float64
 14  agn_sn     158888 non-null  float64
 15  cf_g       158888 non-null  float64
 16  cf_r       158888 non-null  float64
 17  Ha         158888 non-null  float64
 18  Hb         158888 non-null  float64
 19  OII_1      158888 non-null  float64
 20  OII_2      158888 non-null  float64
 21  OIII       158888 non-null  float64
 22  NII        158888 non-null  float64
 23  SII_1      158888 non-null  float64
 24  SII_2      158888 non-null  float64
 25  err_Ha     158888 non-null  float64
 26  err_Hb     158888 non-null  float64
 27  err_OII_1  158888 non-null  float64
 28  err_OII_2  158888 non-null  float64
 29  err_OIII   158888 non-null  float64
 30  err_NII    158888 non-null  float64
 31  err_SII_1  158888 non-null  float64
 32  err_SII_2  158888 non-null  float64
dtypes: float64(33)
memory usage: 40.0 MB
```

In [5]:
```
data_exp.shape # get the shape
```

Out[5]:
```
(158888, 33)
```

In [6]:
```
print(type(data_exp))
```

```
<class 'pandas.core.frame.DataFrame'>
```

In [7]:
```
data_clean_all=data_exp.dropna() # Choose a new name for the clean data set
```

In [8]:
```
data_clean_all.shape # get shape
```

Out[8]:
```
(56619, 33)
```

In [9]:
```
# select a subset of data and then drop NaNs
data_clean_selected= data_exp[['Ha', 'Hb', 'OII_1','NII','SII_2', 'tar_1']].d
print (data_clean_selected)
```

```
              Ha         Hb     OII_1        NII     SII_2    tar_1
3        514.525   180.6580   268.702   139.3550   73.0610  8.86975
5       1011.950   355.5870   494.637   470.7900  124.1400  9.10778
6        284.017    99.6942   120.660   119.6000   47.5061  9.11417
8        542.953   190.6650   112.974   213.9970   69.0888  9.08641
11      1176.250   413.0610   520.896   299.1270  158.9760  8.86924
...          ...        ...       ...        ...       ...      ...
158879  3185.680  1118.5100  2230.150   522.5390  359.2150  8.65184
158880   311.140   109.2260   205.794    42.4103   38.3080  8.80593
158882  3832.080  1346.6800  1474.790  1308.8700  408.8190  9.00067
158883   687.306   241.4110   213.749   202.9140   85.3614  8.94060
158887   321.273   112.7690   150.629   102.4770   41.4276  8.92745

[75852 rows x 6 columns]
```

To utilize supervised methods like classification or regression, you can split the data into input and target sets.

In [10]:
```
input_data= data_clean_selected[['Ha', 'Hb', 'OII_1','NII','SII_2']].values
input_data.shape
```

Out[10]:
```
(75852, 5)
```

In [11]:
```
print (len(input_data))
```

```
75852
```

In [12]:
```
target_data= data_clean_selected[['tar_1']].values
target_data.shape
```

Out[12]:
```
(75852, 1)
```

In [13]:
```
print(type(input_data))
```

```
<class 'numpy.ndarray'>
```

In [14]:
```
import numpy as np # to work with numpy files
```

In [15]:
```
input_data # just vlaues
```

Out[15]:
```
array([[ 514.525 ,  180.658 ,  268.702 ,  139.355 ,   73.061 ],
       [1011.95  ,  355.587 ,  494.637 ,  470.79  ,  124.14  ],
       [ 284.017 ,   99.6942,  120.66  ,  119.6   ,   47.5061],
       ...,
       [3832.08  , 1346.68  , 1474.79  , 1308.87  ,  408.819 ],
       [ 687.306 ,  241.411 ,  213.749 ,  202.914 ,   85.3614],
       [ 321.273 ,  112.769 ,  150.629 ,  102.477 ,   41.4276]])
```

In [16]:
```
input_data[:,1] # the content of the second column
```

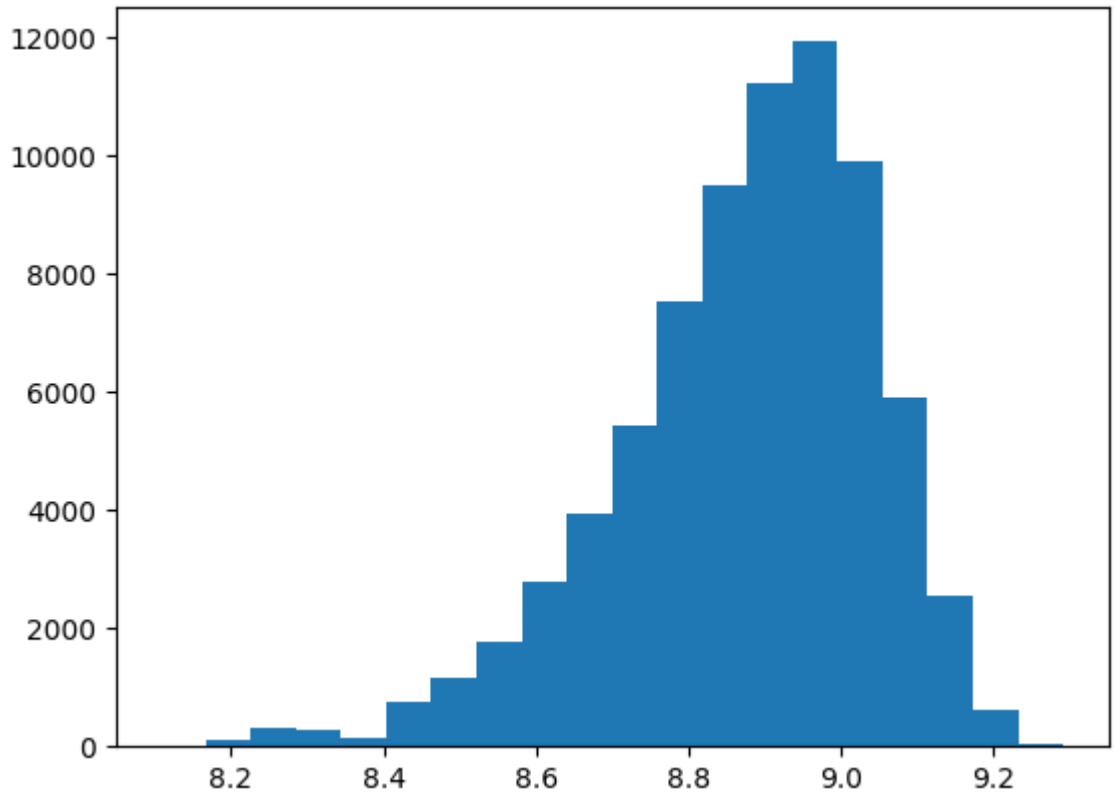
Out[16]:
```
array([ 180.658 ,  355.587 ,   99.6942, ..., 1346.68  ,  241.411 ,
        112.769 ])
```

In [17]:
```
target_data[:,0] # the content of the first column of the target data
```

Out[17]:
```
array([8.86975, 9.10778, 9.11417, ..., 9.00067, 8.9406 , 8.92745])
```

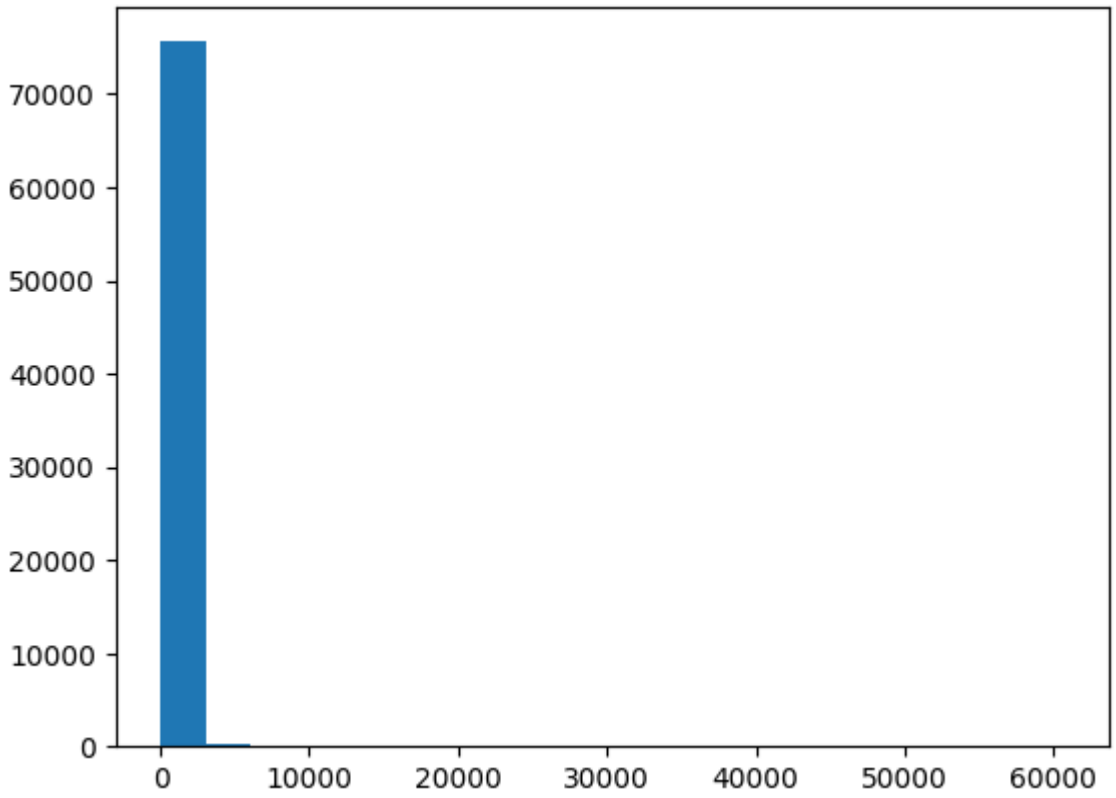
In [18]:
```
# histogram can give us for information
from matplotlib import pyplot as plt
plt.hist(target_data[:,0], bins=20)
```

Out[18]:
```
(array([   23.,   104.,   329.,   267.,   160.,   761.,  1146.,  1781.,
         2774.,  3942.,  5419.,  7531.,  9507., 11204., 11919.,  9880.,
         5914.,  2539.,   609.,    43.]),
 array([8.10717  , 8.1664485, 8.225727 , 8.2850055, 8.344284 , 8.4035625,
        8.462841 , 8.5221195, 8.581398 , 8.6406765, 8.699955 , 8.7592335,
        8.818512 , 8.8777905, 8.937069 , 8.9963475, 9.055626 , 9.1149045,
        9.174183 , 9.2334615, 9.29274  ]),
 <BarContainer object of 20 artists>)
```



In [19]:
```
plt.hist(input_data[:,1],bins=20) # the distribution of the second column
print (len(input_data[:,1])) # the number of samples
```

```
75852
```



In [20]:
```
print ('max= ', np.max(input_data[:,1])) # maximum of the column
print ('min= ',np.min(input_data[:,1])) # maximum of the column
print ('mean= ',np.mean(input_data[:,1])) # maximum of the column
print ('median= ', np.median(input_data[:,1])) # maximum of the column
```

```
max=  60645.8
min=  14.4689
mean=  348.8803033565364
median=  226.672
```

Measures of Central Tendency: These are ways to describe the center of a data set. Commonly used measures include:

Mean: The average of all data points. Median: The middle value in a sorted dataset. Mode: The most frequently occurring value in the dataset. Measures of Dispersion or Variability: These are ways to describe the spread of a data set. Commonly used measures include:

Range: The difference between the maximum and minimum values in the dataset. Variance: The average of the squared differences from the mean. Standard Deviation: The square root of the variance. It is a measure of the amount of variation or dispersion in a set of values.

Interquartile Range (IQR): The range between the first quartile (25th percentile) and the third quartile (75th percentile). Measures of Position: These tell you where a specific data point is within the data set. Common measures include percentiles and quartiles.

Measures of Shape: These describe the shape of the data distribution.

Skewness: Measures the asymmetry of the data. Kurtosis: Measures the "tailedness" of the data distribution. Descriptive statistics are often presented in tables, histograms, bar charts, pie charts, frequency distributions, and other graphical or pictorial methods to visualize the

In [21]:
```
# descriptive analysis
data_exp.describe()
```

Out[21]:

	tar_1	tar_2	tar_3	tar_4	tar_5	tar_6
count	75852.000000	158852.000000	92991.000000	94951.000000	97650.000000	59447.000000
mean	8.874875	7.707753	8.546300	8.646865	8.643520	8.897636
std	0.169721	11.217666	0.241618	0.100803	0.106223	0.168795
min	8.107170	-99.900000	7.762050	7.962620	7.735390	8.352340
25%	8.778530	8.744627	8.379140	8.593565	8.587332	8.779070
50%	8.900750	8.941810	8.572460	8.665100	8.666935	8.916170
75%	8.996460	9.049900	8.734620	8.715075	8.722340	9.030810
max	9.292740	9.466640	9.049880	8.849830	8.843860	9.278110

8 rows × 33 columns

Descriptive statistics

1- Measures of Central Tendency: These are ways to describe the center of a data set. Commonly used measures include:

```
Mean: The average of all data points.
Median: The middle value in a sorted dataset.
Mode: The most frequently occurring value in the dataset.
```

2- Measures of Dispersion or Variability: These are ways to describe the spread of a data set. Commonly used measures include:

```
Range: The difference between the maximum and minimum values in the
dataset.
Variance: The average of the squared differences from the mean.
Standard Deviation: The square root of the variance. It is a measure
of the amount of variation or dispersion in a set of values.
Interquartile Range (IQR): The range between the first quartile (25t
h percentile) and the third quartile (75th percentile).
```

3- Measures of Position: These tell you where a specific data point is within the data set. Common measures include percentiles and quartiles.

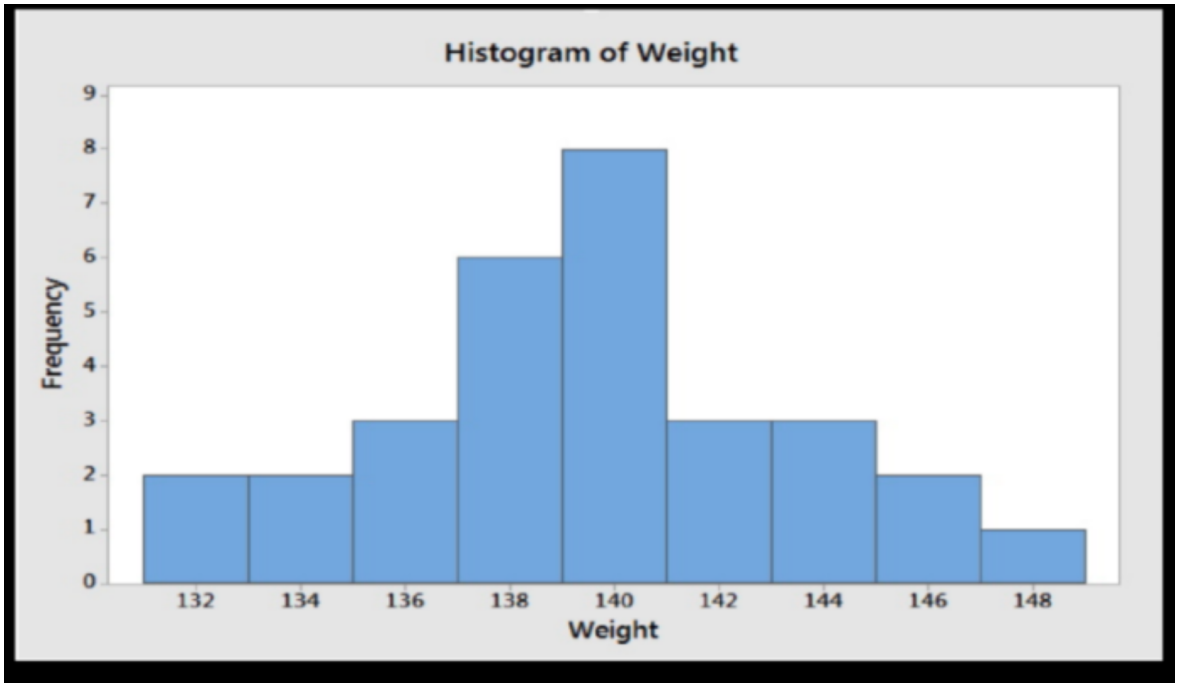
4- Measures of Shape: These describe the shape of the data distribution.

```
Skewness: Measures the asymmetry of the data.
Kurtosis: Measures the "tailedness" of the data distribution.
```

Descriptive statistics are often presented in tables, histograms, bar charts, pie charts, frequency distributions, and other graphical or pictorial methods to visualize the data. They provide a foundational understanding for a dataset before proceeding to more complex statistical analyses or predictive modeling.

How to remove outliers?

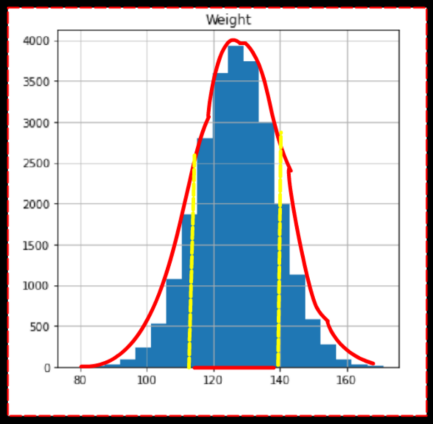
A little about distributions and the information theory



$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$$

mean= 127.07

sigma= 11.66



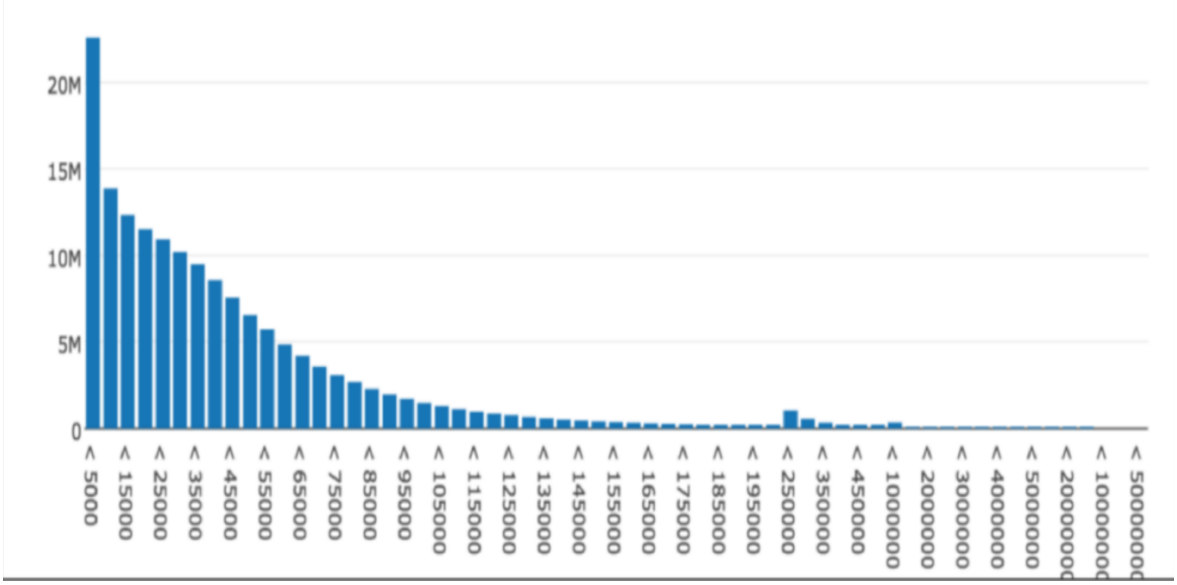
---------- **For Height and Weight file:**

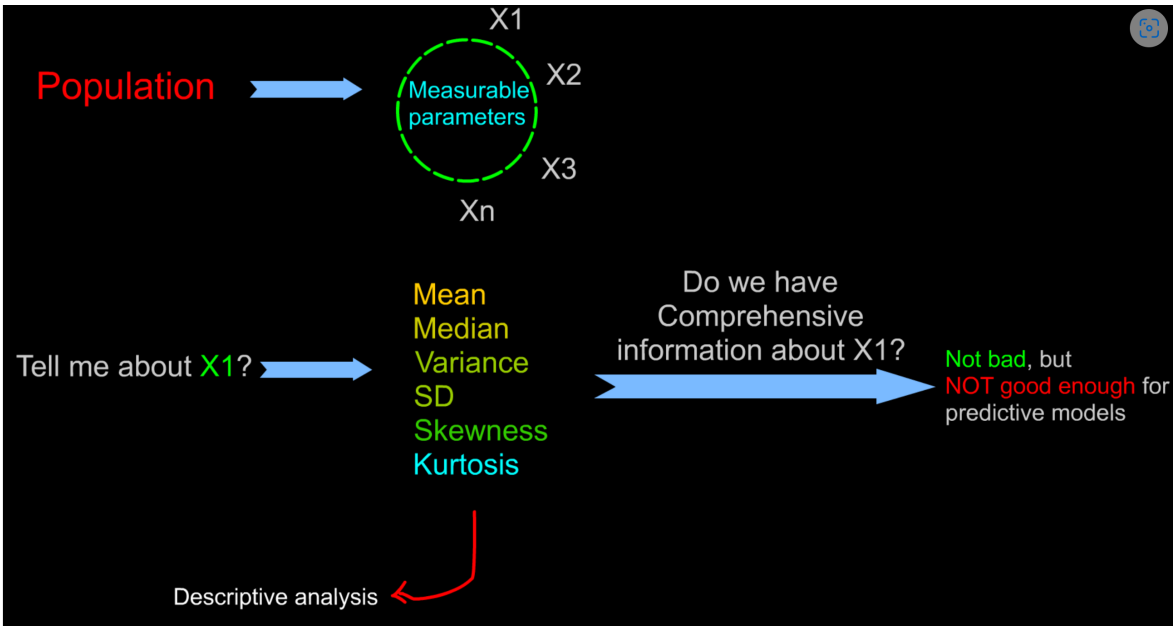
choose a feature (weight)

$X_1 =$ [165, 189, 200, 130, ...]

find μ_0 (mean) and σ_0 (standard diviation)

$$g(x) = \frac{1}{\sigma_0\sqrt{2\pi}} \exp\left[-\frac{1}{2}\frac{(X-\mu_0)^2}{\sigma_0^2}\right]$$





HT: Draw an example of outliers; how to make a decision to remove them

what is the difference between ML, statistics and data science

In [22]:
```
lst =[0, 2,3,4 ,8, 1000, 11]
lst = np.array(lst)
```

In [23]:
```
idx = np.where( (lst<900) & (lst>0) )[0]
print (idx)
```

```
[1 2 3 4 6]
```

In [24]:
```
lst_new = lst[idx]
```

In [25]:
```
lst_new
```

Out[25]:
```
array([ 2,  3,  4,  8, 11])
```

Percentile by Numpy

In [26]:
```
perc_inp= np.percentile(input_data,90, axis = 0)
print (perc_inp)
```

```
[2046.804   718.9598  891.0657  688.1343  237.9669]
```

In [27]:
```
perc_inp[0]
```

Out[27]:
```
2046.8040000000015
```

In [ ]:
```
```

Draw to show the case regarding input_data and target_data

In [28]:
```
idx_good= np.where ( (input_data[:,0]< perc_inp[0]) & (input_data[:,0]>0) &
                     (input_data[:,1]< perc_inp[1]) & (input_data[:,1]>0) &
                     (input_data[:,2]< perc_inp[2]) & (input_data[:,2]>0) &
                     (input_data[:,3]< perc_inp[3]) & (input_data[:,3]>0) &
                     (input_data[:,4]< perc_inp[4]) & (input_data[:,4]>0) )[0
len(idx_good)
```

Out[28]:
```
64486
```

In [29]:
```
input_good = input_data[idx_good]

target_good = target_data[idx_good]

print('size of input_good= ', np.shape(input_good))
print('size of target_good=', np.shape(target_good))
```

```
size of input_good=  (64486, 5)
size of target_good= (64486, 1)
```

>>>>>> The class activity (number 2)

Remove ouliers for percentile= 95 and create new input_data and target_data without ouliers and 0 for all cloumns

speliting input_data and target data to the training set (70%) and the validation set (30%)

In [30]:
```
from sklearn.model_selection import train_test_split # importing the function
```

In [31]:
```
X_train, X_test, y_train, y_test = train_test_split( input_good, target_good,

print ('X_train= ',np.shape(X_train))
print ('y_train= ',np.shape(y_train))
print ('X_test= ',np.shape(X_test))
print ('y_test= ',np.shape(y_test))
```

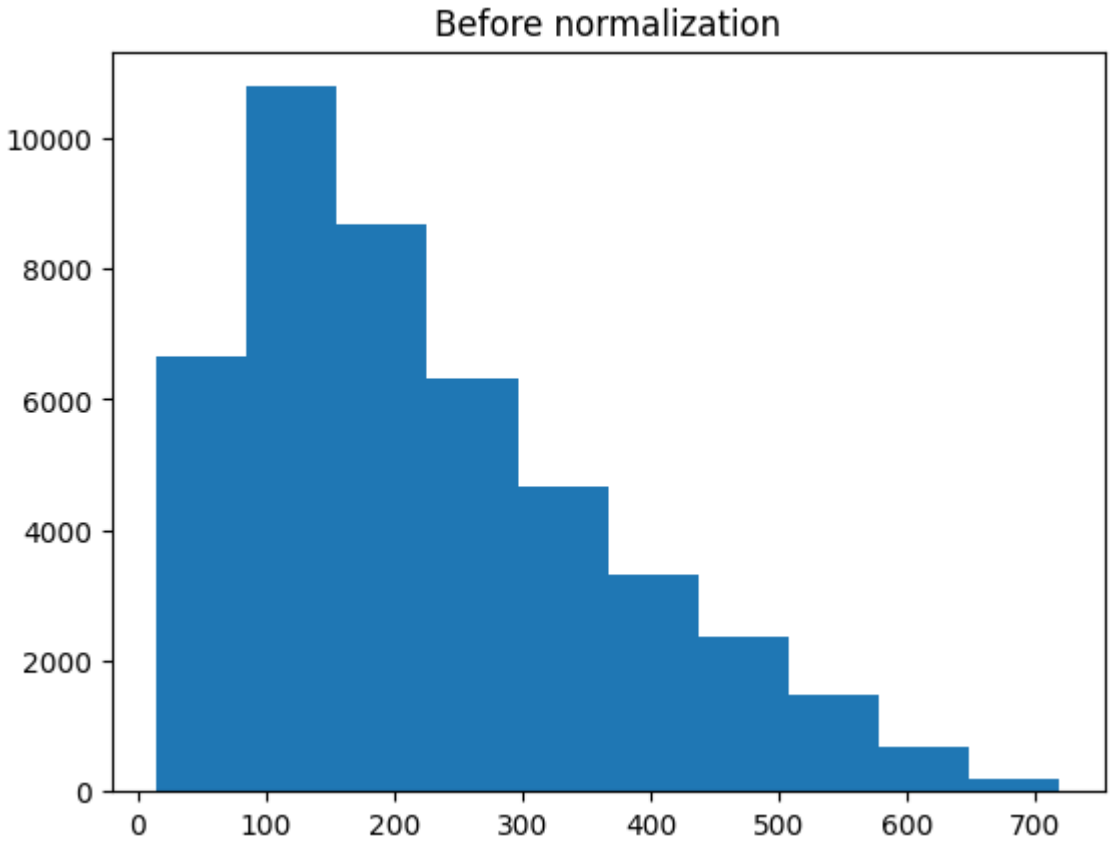
```
X_train=  (45140, 5)
y_train=  (45140, 1)
X_test=  (19346, 5)
y_test=  (19346, 1)
```

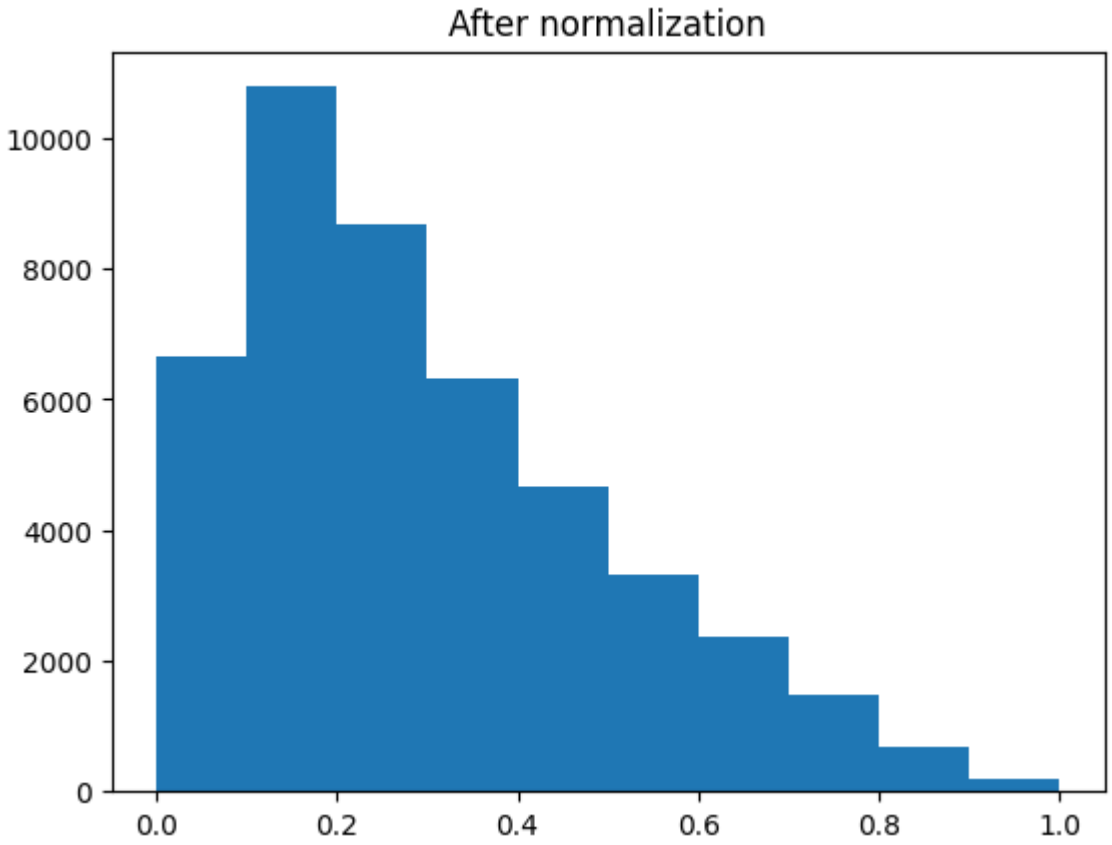
In [32]:
```
# we want to normalize the input using SKlearn

# import two models "StandardScaler" and "MinMaxScaler"
from sklearn.preprocessing import StandardScaler, MinMaxScaler  #line #1

# choose one model with proper name
scaler_MinMax=  MinMaxScaler()

# Find the trainable paramters of the model

scaler_MinMax.fit(X_train)


X_train_MinMax= scaler_MinMax.transform(X_train)  # Line #4

X_test_MinMax= scaler_MinMax.transform(X_test)  # Line #4

print('Done!')
```

```
Done!
```

In [33]:
```
plt.figure(1)
plt.hist(X_train[:,1])
plt.title('Before normalization')

plt.figure(2)
plt.hist(X_train_MinMax[:,1])
plt.title('After normalization')
```

Out[33]:
```
Text(0.5, 1.0, 'After normalization')
```





>>>>>> Activity #3

Choose weight-height file and split the data to training (75%) and test set(25%) and normalize the split data

In [ ]: