Preprocessing: Handling missing data:

In [183…
```
# Read the file IBM-313 Marks - mis.xlsx attached in the mail
import pandas as pd
table1=pd.read_excel(r'C:\Users\lenovo\Downloads\IBM-313 Marks - mis.xlsx')
```

In [184…
```
table1
```

Out[184]:

	S.No.	midexam	miniproject	total_internal	endexam	total	Grace marks
0	1	NaN	20	NaN	12.0	NaN	NaN
1	2	11.05	20	31.05	26.0	57.05	NaN
2	3	NaN	20	NaN	14.0	NaN	NaN
3	4	6.00	10	16.00	13.0	29.00	NaN
4	5	11.35	20	31.35	17.0	48.35	NaN
...	...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.0	42.05	NaN
75	76	12.25	10	22.25	28.0	50.25	NaN
76	77	1.75	10	11.75	NaN	0.00	NaN
77	78	3.00	10	13.00	NaN	0.00	NaN
78	79	5.80	10	15.80	12.0	27.80	NaN

79 rows × 7 columns

Filling NaNs with some value

In [185…
```
#Fill all the NaN in "total" column with 5,
table1['total_internal'].fillna(5, inplace=False)
```

Out[185]:
```
0      5.00
1     31.05
2      5.00
3     16.00
4     31.35
      ...
74    22.05
75    22.25
76    11.75
77    13.00
78    15.80
Name: total_internal, Length: 79, dtype: float64
```

Both inplace= true and inplace = False are used to do some operation on the data but: When inplace = True is used, it performs operation on data and nothing is returned. When inplace=False is used, it performs operation on data and returns a new copy of data.

By default, the inplace parameter in the fillna() method in pandas is set to False. This means that the method returns a new DataFrame or Series with the missing values filled, leaving the original data unchanged unless you explicitly set inplace=True. If you set inplace=True, the operation will modify the original DataFrame or Series directly and return None.

In [186…
```
# we see the original table1 data is same as we had used with inplace=False
#in the above command
table1
```

Out[186]:

	S.No.	midexam	miniproject	total_internal	endexam	total	Grace marks
0	1	NaN	20	NaN	12.0	NaN	NaN
1	2	11.05	20	31.05	26.0	57.05	NaN
2	3	NaN	20	NaN	14.0	NaN	NaN
3	4	6.00	10	16.00	13.0	29.00	NaN
4	5	11.35	20	31.35	17.0	48.35	NaN
...	...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.0	42.05	NaN
75	76	12.25	10	22.25	28.0	50.25	NaN
76	77	1.75	10	11.75	NaN	0.00	NaN
77	78	3.00	10	13.00	NaN	0.00	NaN
78	79	5.80	10	15.80	12.0	27.80	NaN

79 rows × 7 columns

In [187…
```
# how many nans are present in each column?
table1.isna().sum()
```

Out[187]:
```
S.No.              0
midexam            2
miniproject        0
total_internal     2
endexam            2
total              2
Grace marks       79
dtype: int64
```

In [188…
```
table1['midexam'].isna()
```

Out[188]:
```
0      True
1     False
2      True
3     False
4     False
      ...
74    False
75    False
76    False
77    False
78    False
Name: midexam, Length: 79, dtype: bool
```

In [191…
```
# filling all the NaNs with 0
new_tab=table1.fillna(0)
new_tab
```

Out[191]:

	S.No.	midexam	miniproject	total_internal	endexam	total	Grace marks
0	1	0.00	20	0.00	12.0	0.00	0.0
1	2	11.05	20	31.05	26.0	57.05	0.0
2	3	0.00	20	0.00	14.0	0.00	0.0
3	4	6.00	10	16.00	13.0	29.00	0.0

4	5	11.35	20	31.35	17.0	48.35	0.0
...	...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.0	42.05	0.0
75	76	12.25	10	22.25	28.0	50.25	0.0
76	77	1.75	10	11.75	0.0	0.00	0.0
77	78	3.00	10	13.00	0.0	0.00	0.0
78	79	5.80	10	15.80	12.0	27.80	0.0

79 rows × 7 columns

In [192…

```
table1
```

Out[192]:

	S.No.	midexam	miniproject	total_internal	endexam	total	Grace marks
0	1	NaN	20	NaN	12.0	NaN	NaN
1	2	11.05	20	31.05	26.0	57.05	NaN
2	3	NaN	20	NaN	14.0	NaN	NaN
3	4	6.00	10	16.00	13.0	29.00	NaN
4	5	11.35	20	31.35	17.0	48.35	NaN
...	...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.0	42.05	NaN
75	76	12.25	10	22.25	28.0	50.25	NaN
76	77	1.75	10	11.75	NaN	0.00	NaN
77	78	3.00	10	13.00	NaN	0.00	NaN
78	79	5.80	10	15.80	12.0	27.80	NaN

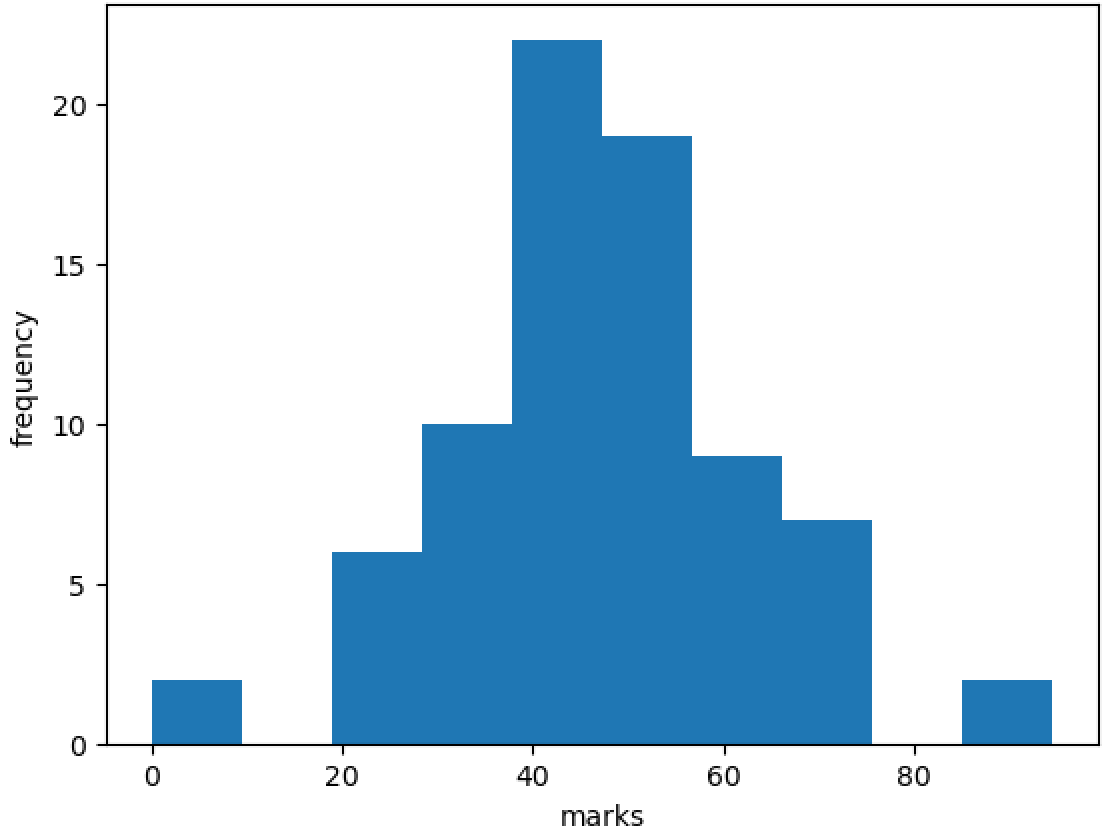
79 rows × 7 columns

In [120…

```
# plot the histogram for total column
import matplotlib.pyplot as plt
plt.hist(table1['total'])
plt.xlabel('marks')
plt.ylabel('frequency')
```

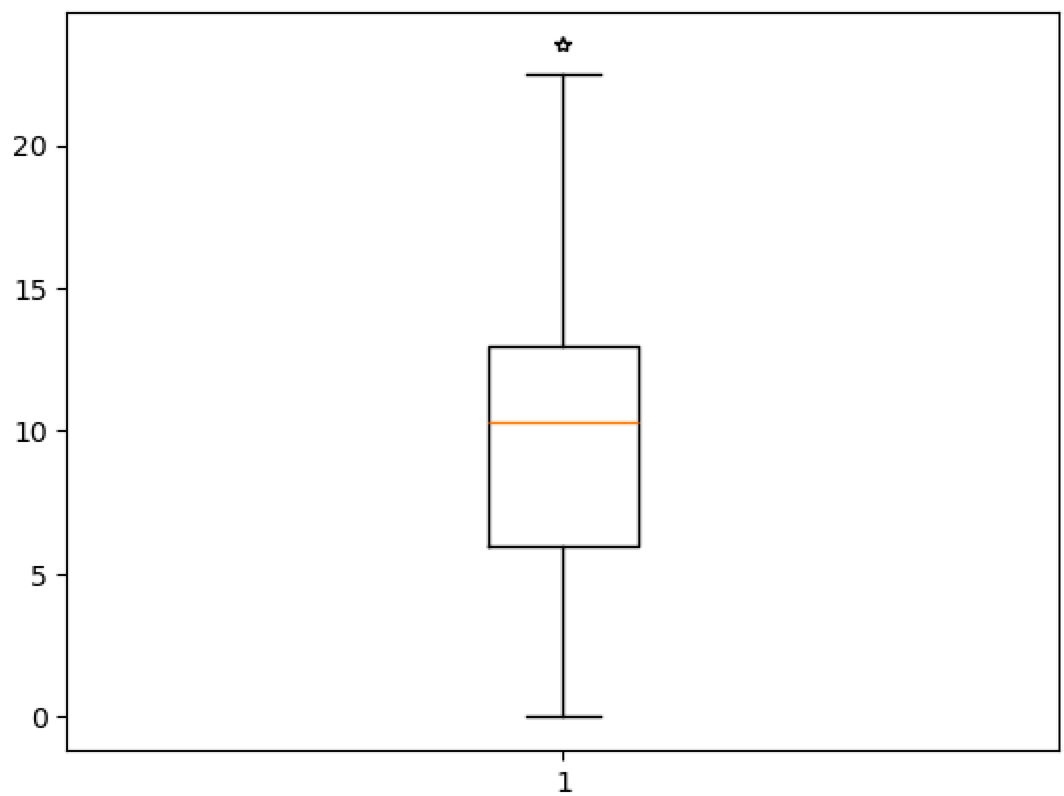
Out[120]:

```
Text(0, 0.5, 'frequency')
```



In [121…

```
# plot the box plot for tall the columns
from matplotlib import pyplot as plt
plt.boxplot(new_tab['midexam'], sym='*')
plt.show()
```



Drop a column having NaNs

In [193…

```
# drop grace marks column as it has all NaNs
#from IBM-313 Marks - mis.xlsx dataset
```

```
# axis =1 signify column
table1.drop(['Grace marks'], axis = 1, inplace=True)
```

In [194…

```
table1
```

Out[194]:

	S.No.	midexam	miniproject	total_internal	endexam	total
0	1	NaN	20	NaN	12.0	NaN
1	2	11.05	20	31.05	26.0	57.05
2	3	NaN	20	NaN	14.0	NaN
3	4	6.00	10	16.00	13.0	29.00
4	5	11.35	20	31.35	17.0	48.35
...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.0	42.05
75	76	12.25	10	22.25	28.0	50.25
76	77	1.75	10	11.75	NaN	0.00
77	78	3.00	10	13.00	NaN	0.00
78	79	5.80	10	15.80	12.0	27.80

79 rows × 6 columns

Drop a row with NaNs

In [195…

```
# drop a particular row with an index suppose 0 and axis 0 (means row)
table1.drop([0], axis = 0, inplace=True)
table1
```

Out[195]:

	S.No.	midexam	miniproject	total_internal	endexam	total
1	2	11.05	20	31.05	26.0	57.05
2	3	NaN	20	NaN	14.0	NaN
3	4	6.00	10	16.00	13.0	29.00
4	5	11.35	20	31.35	17.0	48.35
5	6	11.00	20	31.00	24.0	55.00
...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.0	42.05
75	76	12.25	10	22.25	28.0	50.25
76	77	1.75	10	11.75	NaN	0.00
77	78	3.00	10	13.00	NaN	0.00
78	79	5.80	10	15.80	12.0	27.80

78 rows × 6 columns

In [196…

```
# get the description of the dataset
table1.describe()
```

Out[196]:

	S.No.	midexam	miniproject	total_internal	endexam	total
count	78.000000	77.000000	78.000000	77.000000	76.000000	77.000000

mean	40.500000	10.272727	16.512821	26.740260	21.134868	47.097403
std	22.660538	4.984968	4.916824	8.612434	8.077277	16.477932
min	2.000000	0.700000	10.000000	11.200000	7.000000	0.000000
25%	21.250000	7.000000	10.500000	19.500000	17.000000	38.000000
50%	40.500000	10.500000	15.000000	27.500000	20.000000	45.500000
75%	59.750000	13.050000	22.000000	33.500000	24.000000	55.400000
max	79.000000	23.500000	22.000000	45.500000	50.000000	94.500000

Fill the NaNs in endexam by mean endexam

In [197…

```
# fill the NaNs in endexam by mean
table1['endexam'].fillna(table1['endexam'].mean(), inplace=True)
table1
```

Out[197]:

	S.No.	midexam	miniproject	total_internal	endexam	total
1	2	11.05	20	31.05	26.000000	57.05
2	3	NaN	20	NaN	14.000000	NaN
3	4	6.00	10	16.00	13.000000	29.00
4	5	11.35	20	31.35	17.000000	48.35
5	6	11.00	20	31.00	24.000000	55.00
...	...	...	...	...	...	...
74	75	12.05	10	22.05	20.000000	42.05
75	76	12.25	10	22.25	28.000000	50.25
76	77	1.75	10	11.75	21.134868	0.00
77	78	3.00	10	13.00	21.134868	0.00
78	79	5.80	10	15.80	12.000000	27.80

78 rows × 6 columns

More on handling missing values

In [128…

```
# Create your own txt file as shown below (df)
import pandas as pd
df=pd.read_csv(r'C:\Users\lenovo\Downloads\mydata.txt')
```

In [129…

```
df
```

Out[129]:

	A	B	C	D
0	1	2	3.0	4.0
1	5	6	NaN	8.0
2	9	10	12.0	NaN

In [130…

```
# Count the missing values per column
df.isnull().sum()
```

Out[130]:

```
A     0
B     0
```

```
C    1
D    1
dtype: int64
```

In pandas, df.values is an attribute that returns the data of the DataFrame df as a NumPy array. This array contains the underlying data of the DataFrame without the index or column labels.

In [131…
```
df.values
```

Out[131]:
```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6., nan,  8.],
       [ 9., 10., 12., nan]])
```

In [132…
```
df.ndim
```

Out[132]:
```
2
```

Eliminating training examples with missing values

One of the easiest ways to deal with missing data is to remove the corresponding features (columns) or training examples(rows) from the entire dataset. Rows with missing values can easily be dropped via dropna method

In [133…
```
df.dropna(axis=0)
```

Out[133]:

	A	B	C	D
0	1	2	3.0	4.0

Similarly we can drop columns that have at least one NaN in any row by setting the axis argument to 1

In [134…
```
df.dropna(axis=1)
```

Out[134]:

	A	B
0	1	2
1	5	6
2	9	10

In [135…
```
df
```

Out[135]:

	A	B	C	D
0	1	2	3.0	4.0
1	5	6	NaN	8.0
2	9	10	12.0	NaN

In [136…
```
# Drop rows that have fewer than 4 real values
df.dropna(thresh=4)
```

Out[136]:

	A	B	C	D
0	1	2	3.0	4.0

thresh=4: This parameter specifies that a row must have at least 4 non-NaN values to be retained in the DataFrame. If a row has fewer than 4 non-NaN values, it will be dropped.

In [137…
```
df
```

Out[137]:

	A	B	C	D
0	1	2	3.0	4.0
1	5	6	NaN	8.0
2	9	10	12.0	NaN

Imputing missing values

In [138…
```
from sklearn.impute import SimpleImputer
import numpy as np
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')
# fit_transform() preprocess the data model training
X= imputer.fit_transform(df)
X
```

Out[138]:
```
array([[ 1. ,  2. ,  3. ,  4. ],
       [ 5. ,  6. ,  7.5,  8. ],
       [ 9. , 10. , 12. ,  6. ]])
```

We can acheive the same mean imputation directly in Dataframe object via:

In [139…
```
df.fillna(df.mean())
```

Out[139]:

	A	B	C	D
0	1	2	3.0	4.0
1	5	6	7.5	8.0
2	9	10	12.0	6.0

Handling Categorical data (Categorical data encoding with pandas)

In [175…
```
import pandas as pd
df = pd.DataFrame([['green', 'M', 10.1, 'class2'],
                   ['red', 'L',13.5,'class1'],
                  ['blue', 'XL', 15.3, 'class2']])
df.columns = ['color', 'size', 'price', 'classlabel']
df
```

Out[175]:

	color	size	price	classlabel
0	green	M	10.1	class2
1	red	L	13.5	class1
2	blue	XL	15.3	class2

We can see above that the dataframe contains a nominal feature (color) and an odinal feature (size)

Mapping ordinal features

In [176…
```
# mapping ordinal features
size_mapping = {'XL':3,
```

```
                 'L':2,
                 'M':1}
df['size']=df['size'].map(size_mapping)
df
```

Out[176]:

	color	size	price	classlabel
0	green	1	10.1	class2
1	red	2	13.5	class1
2	blue	3	15.3	class2

Tranform integer value back to original string reprsentaion by applying reverse mapping dictionary

In [177…

```
# transform integer back to original string by defining reverse mapping dictionary
inv_size_mapping = {v: k for k, v in size_mapping.items()}
df['size'].map(inv_size_mapping)
```

Out[177]:

```
0     M
1     L
2    XL
Name: size, dtype: object
```

Encoding class labels

In [143…

```
import numpy as np
class_mapping = {label:idx for idx, label in
                 enumerate(np.unique(df['classlabel']))}
class_mapping
```

Out[143]:

```
{'class1': 0, 'class2': 1}
```

In [144…

```
# we can use the mapping dictionary to trasnform the class labels into integers
df['classlabel'] = df['classlabel'].map(class_mapping)
df
```

Out[144]:

	color	size	price	classlabel
0	green	1	10.1	1
1	red	2	13.5	0
2	blue	3	15.3	1

We can reverse the key-value pairs in the mapping dictionary as follows to map the converted class labels back to the orignal string reprsentaions

In [145…

```
inv_class_mapping = {v:k for k, v in class_mapping.items()}
df['classslabel']=df['classlabel'].map(inv_class_mapping)
df
```

Out[145]:

	color	size	price	classlabel	classslabel
0	green	1	10.1	1	class2
1	red	2	13.5	0	class1
2	blue	3	15.3	1	class2

In [199…

```
# Alternatively, convenient is to use LabelEncoder class
```

```
from sklearn.preprocessing import LabelEncoder
class_le = LabelEncoder()
y = class_le.fit_transform(df['classlabel'].values)
y
```

Out[199]:
```
array([1, 0, 1])
```

We can use inverse_tranform method to tranform the integer class labels back into orogimal string reprsentaion

In [200…
```
class_le.inverse_transform(y)
```

Out[200]:
```
array(['class2', 'class1', 'class2'], dtype=object)
```

Perform one hot encoding on nominal features

In [147…
```
X = df[['color','size','price']].values
X
```

Out[147]:
```
array([['green', 1, 10.1],
       ['red', 2, 13.5],
       ['blue', 3, 15.3]], dtype=object)
```

In [148…
```
color_le = LabelEncoder()
X[:,0] = color_le.fit_transform(X[:,0])
X
```

Out[148]:
```
array([[1, 1, 10.1],
       [2, 2, 13.5],
       [0, 3, 15.3]], dtype=object)
```

After executing the preceeding code, the first column of the NumPy array X, now holds the new color values, which are encoded as : blue=0, green=1, and red=2.

In [206…
```
from sklearn.preprocessing import OneHotEncoder
X = df[['color', 'size', 'price']].values
color_ohe = OneHotEncoder()
color_ohe.fit_transform(X[:, 0].reshape(-1,1)).toarray()
```

Out[206]:
```
array([[0., 1., 0.],
       [0., 0., 1.],
       [1., 0., 0.]])
```

In [150…
```
from sklearn.compose import ColumnTransformer
X = df[['color', 'size', 'price']].values
c_transf = ColumnTransformer([('onehot', OneHotEncoder(), [0]), ('nothing', 'passthrough
c_transf.fit_transform(X)
```

Out[150]:
```
array([[0.0, 1.0, 0.0, 1, 10.1],
       [0.0, 0.0, 1.0, 2, 13.5],
       [1.0, 0.0, 0.0, 3, 15.3]], dtype=object)
```

In the above code we specified that we want to modify only first column and leave the other 2 columns untouched via passthrough argument.

In [151…
```
pd.get_dummies(df[['price', 'color', 'size']])
```

Out[151]:

	price	size	color_blue	color_green	color_red
0	10.1	1	False	True	False
1	13.5	2	False	False	True

2	15.3	3	True	False	False

In [152…

```
pd.get_dummies(df[['price', 'color', 'size']], drop_first= True)
```

Out[152]:

	price	size	color_green	color_red
0	10.1	1	True	False
1	13.5	2	False	True
2	15.3	3	False	False

import another dataset called data encoding.csv

In [153…

```
#Data encoding: use dataset data encoding.csv aatached in mail
#encoding
import pandas as pd
df=pd.read_csv(r'C:\Users\lenovo\Downloads\data encoding (1).csv')
df.head(5)
```

Out[153]:

	Roll_no	gender	marks	placed
0	100	M	55	y
1	200	F	67	y
2	300	M	67	y
3	400	F	12	n
4	500	M	90	y

In [154…

```
from sklearn.preprocessing import LabelEncoder
label_1 = LabelEncoder()
df['gender']=label_1.fit_transform(df['gender'])
```

In [155…

```
df
```

Out[155]:

	Roll_no	gender	marks	placed
0	100	1	55	y
1	200	0	67	y
2	300	1	67	y
3	400	0	12	n
4	500	1	90	y
5	600	0	56	n
6	700	1	90	y

In [156…

```
df['placed']=label_1.fit_transform(df['placed'])
df
```

Out[156]:

	Roll_no	gender	marks	placed
0	100	1	55	1
1	200	0	67	1
2	300	1	67	1
3	400	0	12	0

4	500	1	90	1
5	600	0	56	0
6	700	1	90	1

In [157…

```
# Normalization of data using MinMaxScaler(map between 0 to 1):use Wine.csv as attached
import pandas as pd
df=pd.read_csv(r'C:\Users\lenovo\Downloads\wine.csv')
df.head(5)
```

Out[157]:

	Wine	Alcohol	Malic.acid	Ash	Acl	Mg	Phenols	Flavanoids	Nonflavanoid.phenols	Proanth	Color.int
0	1	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29	5.64
1	1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28	4.38
2	1	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81	5.68
3	1	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18	7.80
4	1	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82	4.32

In [158…

```
from sklearn.preprocessing import MinMaxScaler
scaling = MinMaxScaler()
df[["Alcohol", "Malic.acid"]]=scaling.fit_transform \
(df[["Alcohol","Malic.acid"]])
```

In [159…

```
df[["Alcohol", "Malic.acid"]]
```

Out[159]:

	Alcohol	Malic.acid
0	0.842105	0.191700
1	0.571053	0.205534
2	0.560526	0.320158
3	0.878947	0.239130
4	0.581579	0.365613
...	...	...
173	0.705263	0.970356
174	0.623684	0.626482
175	0.589474	0.699605
176	0.563158	0.365613
177	0.815789	0.664032

178 rows × 2 columns

In [160…

```
# standardization of data
from sklearn.preprocessing import StandardScaler
import numpy as np
scaler = StandardScaler()
scaled_data = scaler.fit_transform(df[['Acl','Mg']])
```

In [161…

```
scaled_data
```

Out[161]:

```
array([[-1.16959318e+00,  1.91390522e+00],
       [-2.49084714e+00,  1.81450206e-02],
       [-2.68738198e-01,  8.83583612e-02],
       [-8.09251184e-01,  9.30918449e-01],
```

```
[ 4.51945783e-01,  1.28198515e+00],
[-1.28970717e+00,  8.60705108e-01],
[-1.46987817e+00, -2.62708342e-01],
[-5.69023190e-01,  1.49262517e+00],
[-1.65004916e+00, -1.92495001e-01],
[-1.04947918e+00, -1.22281661e-01],
[-4.48909194e-01,  3.69211724e-01],
[-8.09251184e-01, -3.32921683e-01],
[-1.04947918e+00, -7.54201726e-01],
[-2.43079014e+00, -6.13775045e-01],
[-2.25061915e+00,  1.58571702e-01],
[-6.89137187e-01,  8.60705108e-01],
[ 1.51660791e-01,  1.42241183e+00],
[ 1.51660791e-01,  1.07134513e+00],
[-8.99336682e-01,  5.79851746e-01],
[-1.28970717e+00,  1.14155847e+00],
[-1.04947918e+00,  1.84369188e+00],
[-2.68738198e-01,  1.58571702e-01],
[-8.69308183e-01,  8.83583612e-02],
[-5.08966192e-01, -3.32921683e-01],
[ 1.51660791e-01, -2.62708342e-01],
[ 1.65308575e+00,  1.70326520e+00],
[-1.01945068e+00, -4.73348364e-01],
[-7.49194186e-01, -4.03135023e-01],
[-2.85102043e-02,  5.09638405e-01],
[-1.04947918e+00, -2.62708342e-01],
[ 9.02373272e-01,  8.83583612e-02],
[-1.18595702e-01,  4.39425064e-01],
[-6.89137187e-01,  2.98998383e-01],
[ 1.51829490e-03,  2.26497192e+00],
[-1.48624201e-01,  7.20278427e-01],
[ 3.01803287e-01,  1.81450206e-02],
[-1.19962167e+00,  7.20278427e-01],
[-4.48909194e-01, -1.22281661e-01],
[-1.19962167e+00, -1.22281661e-01],
[-1.89027716e+00,  1.98411856e+00],
[-9.89422180e-01,  1.21177181e+00],
[-2.08681200e-01, -6.83988386e-01],
[-1.34976417e+00,  8.83583612e-02],
[-5.99051690e-01,  2.28785042e-01],
[-7.49194186e-01,  5.09638405e-01],
[-1.78652700e-01,  7.90491768e-01],
[-1.04947918e+00,  1.58571702e-01],
[-1.04947918e+00,  8.83583612e-02],
[-2.08681200e-01,  2.28785042e-01],
[-6.29080189e-01,  5.79851746e-01],
[-2.13050515e+00, -5.43561704e-01],
[-6.89137187e-01, -4.03135023e-01],
[-1.65004916e+00,  7.90491768e-01],
[-7.19165687e-01,  1.07134513e+00],
[-9.29365181e-01,  1.28198515e+00],
[ 3.01803287e-01,  1.14155847e+00],
[-9.59393680e-01,  1.28198515e+00],
[-8.09251184e-01,  1.58571702e-01],
[-8.39279684e-01,  5.79851746e-01],
[-2.67101814e+00, -8.24415067e-01],
[-1.04947918e+00,  8.83583612e-02],
[-8.09251184e-01,  1.81450206e-02],
[-4.48909194e-01, -4.03135023e-01],
[-1.48624201e-01, -8.94628408e-01],
[-1.48624201e-01,  2.98998383e-01],
[-4.18880694e-01, -1.22281661e-01],
[-1.34976417e+00, -1.52654847e+00],
[ 3.15467941e-02, -1.52654847e+00],
[-7.49194186e-01,  7.20278427e-01],
[-8.09251184e-01,  3.59902539e+00],
```

```
[ 2.71774788e-01,  2.28785042e-01],
[ 1.65308575e+00, -9.64841748e-01],
[ 1.35280076e+00, -8.94628408e-01],
[ 3.15451071e+00,  2.75646531e+00],
[ 4.51945783e-01,  8.83583612e-02],
[-1.04947918e+00, -1.92495001e-01],
[-1.04947918e+00, -9.64841748e-01],
[-4.48909194e-01,  8.60705108e-01],
[-1.40982117e+00,  2.54582528e+00],
[ 1.05251577e+00,  8.83583612e-02],
[-1.48624201e-01, -9.64841748e-01],
[-2.08681200e-01, -9.64841748e-01],
[ 1.35280076e+00, -1.52654847e+00],
[ 9.02373272e-01, -1.03505509e+00],
[-4.48909194e-01, -4.03135023e-01],
[-4.48909194e-01, -5.20683200e-02],
[ 9.92458769e-01, -6.83988386e-01],
[ 1.95337074e+00, -8.24415067e-01],
[ 6.32116779e-01, -1.10526843e+00],
[ 1.23268676e+00, -2.08825520e+00],
[-2.98766697e-01, -1.31590845e+00],
[ 7.52230776e-01, -9.64841748e-01],
[ 3.61860286e-01, -1.38612179e+00],
[-4.48909194e-01, -8.24415067e-01],
[-4.48909194e-01, -1.22281661e-01],
[-1.48624201e-01,  4.37137214e+00],
[ 6.02088279e-01,  2.40539860e+00],
[-1.04947918e+00, -1.03505509e+00],
[-2.98766697e-01, -8.24415067e-01],
[-4.48909194e-01, -8.24415067e-01],
[-5.99051690e-01, -1.92495001e-01],
[-2.98766697e-01, -8.24415067e-01],
[ 4.51945783e-01, -1.22281661e-01],
[ 1.51829490e-03, -9.64841748e-01],
[ 3.01803287e-01, -1.03505509e+00],
[ 7.52230776e-01, -6.83988386e-01],
[-1.48624201e-01, -1.38612179e+00],
[ 9.02373272e-01, -1.10526843e+00],
[-1.48624201e-01, -5.43561704e-01],
[ 1.51660791e-01, -4.03135023e-01],
[ 1.51829490e-03,  5.09638405e-01],
[ 4.51945783e-01, -8.24415067e-01],
[ 1.51660791e-01,  2.28785042e-01],
[ 4.51945783e-01, -8.24415067e-01],
[ 9.02373272e-01, -1.10526843e+00],
[ 6.02088279e-01, -1.03505509e+00],
[ 3.91888785e-01, -9.64841748e-01],
[ 9.02373272e-01,  5.79851746e-01],
[-1.04947918e+00, -1.38612179e+00],
[-1.48624201e-01, -8.94628408e-01],
[ 1.51660791e-01, -2.62708342e-01],
[ 2.70408323e+00,  1.35219849e+00],
[ 2.10351324e+00,  1.58571702e-01],
[ 6.02088279e-01, -9.64841748e-01],
[ 4.51945783e-01, -1.24569511e+00],
[ 4.51945783e-01, -1.03505509e+00],
[ 6.02088279e-01, -9.64841748e-01],
[ 2.70408323e+00, -5.43561704e-01],
[ 1.50294326e+00, -8.24415067e-01],
[ 7.52230776e-01, -1.38612179e+00],
[-4.48909194e-01,  1.56283851e+00],
[ 1.51660791e-01,  2.98998383e-01],
[ 1.35280076e+00, -1.22281661e-01],
[ 6.02088279e-01,  4.39425064e-01],
[-5.99051690e-01, -1.03505509e+00],
[-2.98766697e-01, -4.03135023e-01],
```

```
       [ 4.51945783e-01, -7.54201726e-01],
       [ 1.65308575e+00, -2.62708342e-01],
       [ 1.51829490e-03, -8.24415067e-01],
       [ 1.35280076e+00,  8.83583612e-02],
       [ 4.51945783e-01, -2.62708342e-01],
       [ 1.51660791e-01, -7.54201726e-01],
       [ 1.20265826e+00, -1.92495001e-01],
       [ 1.51660791e-01, -5.43561704e-01],
       [-2.98766697e-01,  8.60705108e-01],
       [ 4.51945783e-01,  1.58571702e-01],
       [ 1.51660791e-01, -1.38612179e+00],
       [ 6.02088279e-01, -9.64841748e-01],
       [ 6.02088279e-01, -5.43561704e-01],
       [ 6.02088279e-01,  9.30918449e-01],
       [ 1.35280076e+00,  1.63305186e+00],
       [ 7.52230776e-01,  8.60705108e-01],
       [ 1.80322825e+00,  1.14155847e+00],
       [-2.98766697e-01, -1.22281661e-01],
       [ 1.51660791e-01,  2.28785042e-01],
       [ 7.52230776e-01, -4.73348364e-01],
       [ 1.51829490e-03, -7.54201726e-01],
       [ 2.25365574e+00, -1.92495001e-01],
       [ 1.65308575e+00, -1.22281661e-01],
       [ 9.02373272e-01, -7.54201726e-01],
       [ 4.51945783e-01, -8.24415067e-01],
       [ 1.51660791e-01,  5.09638405e-01],
       [ 7.52230776e-01,  4.39425064e-01],
       [-2.98766697e-01,  4.39425064e-01],
       [ 7.52230776e-01, -6.83988386e-01],
       [ 9.02373272e-01, -8.24415067e-01],
       [ 1.05251577e+00,  7.90491768e-01],
       [ 1.51829490e-03, -8.24415067e-01],
       [ 1.50294326e+00,  3.69211724e-01],
       [ 1.65308575e+00,  8.60705108e-01],
       [-1.48624201e-01, -2.62708342e-01],
       [ 1.51829490e-03, -9.64841748e-01],
       [ 1.51660791e-01, -6.13775045e-01],
       [ 3.01803287e-01, -3.32921683e-01],
       [ 1.05251577e+00,  1.58571702e-01],
       [ 1.51660791e-01,  1.42241183e+00],
       [ 1.51660791e-01,  1.42241183e+00],
       [ 1.50294326e+00, -2.62708342e-01]])
```

In [ ]: