```
import pandas as pd
import numpy as np
from sklearn.datasets import load_iris

# Load iris dataset into a DataFrame
df_iris = pd.DataFrame(load_iris().data, columns=load_iris().feature_names)
# Convert DataFrame to NumPy array
iris_array = df_iris.values
# Display the first few rows of the DataFrame
print(df_iris.head())
# Display the NumPy array
print(iris_array)
```

```
      [5.8 2.6 4.  1.2]
      [5.  2.3 3.3 1. ]
      [5.6 2.7 4.2 1.3]
      [5.7 3.  4.2 1.2]
      [5.7 2.9 4.2 1.3]
      [6.2 2.9 4.3 1.3]
      [5.1 2.5 3.  1.1]
      [5.7 2.8 4.1 1.3]
      [6.3 3.3 6.  2.5]
      [5.8 2.7 5.1 1.9]
      [7.1 3.  5.9 2.1]
      [6.3 2.9 5.6 1.8]
      [6.5 3.  5.8 2.2]
      [7.6 3.  6.6 2.1]
      [4.9 2.5 4.5 1.7]
      [7.3 2.9 6.3 1.8]
      [6.7 2.5 5.8 1.8]
      [7.2 3.6 6.1 2.5]
      [6.5 3.2 5.1 2. ]
      [6.4 2.7 5.3 1.9]
      [6.8 3.  5.5 2.1]
      [5.7 2.5 5.  2. ]
      [5.8 2.8 5.1 2.4]
      [6.4 3.2 5.3 2.3]
      [6.5 3.  5.5 1.8]
      [7.7 3.8 6.7 2.2]
      [7.7 2.6 6.9 2.3]
      [6.  2.2 5.  1.5]
      [6.9 3.2 5.7 2.3]
      [5.6 2.8 4.9 2. ]
      [7.7 2.8 6.7 2. ]
      [6.3 2.7 4.9 1.8]
      [6.7 3.3 5.7 2.1]
      [7.2 3.2 6.  1.8]
      [6.2 2.8 4.8 1.8]
      [6.1 3.  4.9 1.8]
      [6.4 2.8 5.6 2.1]
      [7.2 3.  5.8 1.6]
      [7.4 2.8 6.1 1.9]
      [7.9 3.8 6.4 2. ]
      [6.4 2.8 5.6 2.2]
      [6.3 2.8 5.1 1.5]
      [6.1 2.6 5.6 1.4]
      [7.7 3.  6.1 2.3]
      [6.3 3.4 5.6 2.4]
      [6.4 3.1 5.5 1.8]
      [6.  3.  4.8 1.8]
      [6.9 3.1 5.4 2.1]
      [6.7 3.1 5.6 2.4]
      [6.9 3.1 5.1 2.3]
      [5.8 2.7 5.1 1.9]
      [6.8 3.2 5.9 2.3]
      [6.7 3.3 5.7 2.5]
      [6.7 3.  5.2 2.3]
      [6.3 2.5 5.  1.9]
      [6.5 3.  5.2 2. ]
      [6.2 3.4 5.4 2.3]
      [5.9 3.  5.1 1.8]]
```

```
features = np.array(iris_array)
features
```

```
       [6.2, 2.9, 4.3, 1.3],
       [5.1, 2.5, 3. , 1.1],
       [5.7, 2.8, 4.1, 1.3],
       [6.3, 3.3, 6. , 2.5],
       [5.8, 2.7, 5.1, 1.9],
       [7.1, 3. , 5.9, 2.1],
       [6.3, 2.9, 5.6, 1.8],
       [6.5, 3. , 5.8, 2.2],
       [7.6, 3. , 6.6, 2.1],
       [4.9, 2.5, 4.5, 1.7],
       [7.3, 2.9, 6.3, 1.8],
       [6.7, 2.5, 5.8, 1.8],
       [7.2, 3.6, 6.1, 2.5],
       [6.5, 3.2, 5.1, 2. ],
       [6.4, 2.7, 5.3, 1.9],
       [6.8, 3. , 5.5, 2.1],
       [5.7, 2.5, 5. , 2. ],
       [5.8, 2.8, 5.1, 2.4],
       [6.4, 3.2, 5.3, 2.3],
       [6.5, 3. , 5.5, 1.8],
       [7.7, 3.8, 6.7, 2.2],
       [7.7, 2.6, 6.9, 2.3],
       [6. , 2.2, 5. , 1.5],
       [6.9, 3.2, 5.7, 2.3],
       [5.6, 2.8, 4.9, 2. ],
       [7.7, 2.8, 6.7, 2. ],
       [6.3, 2.7, 4.9, 1.8],
       [6.7, 3.3, 5.7, 2.1],
       [7.2, 3.2, 6. , 1.8],
       [6.2, 2.8, 4.8, 1.8],
       [6.1, 3. , 4.9, 1.8],
       [6.4, 2.8, 5.6, 2.1],
       [7.2, 3. , 5.8, 1.6],
       [7.4, 2.8, 6.1, 1.9],
       [7.9, 3.8, 6.4, 2. ],
       [6.4, 2.8, 5.6, 2.2],
       [6.3, 2.8, 5.1, 1.5],
       [6.1, 2.6, 5.6, 1.4],
       [7.7, 3. , 6.1, 2.3],
       [6.3, 3.4, 5.6, 2.4],
       [6.4, 3.1, 5.5, 1.8],
       [6. , 3. , 4.8, 1.8],
       [6.9, 3.1, 5.4, 2.1],
       [6.7, 3.1, 5.6, 2.4],
       [6.9, 3.1, 5.1, 2.3],
       [5.8, 2.7, 5.1, 1.9],
       [6.8, 3.2, 5.9, 2.3],
       [6.7, 3.3, 5.7, 2.5],
       [6.7, 3. , 5.2, 2.3],
       [6.3, 2.5, 5. , 1.9],
       [6.5, 3. , 5.2, 2. ],
       [6.2, 3.4, 5.4, 2.3],
       [5.9, 3. , 5.1, 1.8]])
```

```
import numpy as np
from sklearn import preprocessing
minmax_scaler = preprocessing.MinMaxScaler(feature_range =(0,1))
scaled_feature = minmax_scaler.fit_transform(features)
scaled_feature
```

```
[0.61111111, 0.41666667, 0.76271186, 0.70833333],
[0.94444444, 0.75      , 0.96610169, 0.875     ],
[0.94444444, 0.25      , 1.        , 0.91666667],
[0.47222222, 0.08333333, 0.6779661 , 0.58333333],
[0.72222222, 0.5       , 0.79661017, 0.91666667],
[0.36111111, 0.33333333, 0.66101695, 0.79166667],
[0.94444444, 0.33333333, 0.96610169, 0.79166667],
[0.55555556, 0.29166667, 0.66101695, 0.70833333],
[0.66666667, 0.54166667, 0.79661017, 0.83333333],
[0.80555556, 0.5       , 0.84745763, 0.70833333],
[0.52777778, 0.33333333, 0.6440678 , 0.70833333],
[0.5       , 0.41666667, 0.66101695, 0.70833333],
[0.58333333, 0.33333333, 0.77966102, 0.83333333],
[0.80555556, 0.41666667, 0.81355932, 0.625     ],
[0.86111111, 0.33333333, 0.86440678, 0.75      ],
[1.        , 0.75      , 0.91525424, 0.79166667],
[0.58333333, 0.33333333, 0.77966102, 0.875     ],
[0.55555556, 0.33333333, 0.69491525, 0.58333333],
[0.5       , 0.25      , 0.77966102, 0.54166667],
[0.94444444, 0.41666667, 0.86440678, 0.91666667],
[0.55555556, 0.58333333, 0.77966102, 0.95833333],
[0.58333333, 0.45833333, 0.76271186, 0.70833333],
[0.47222222, 0.41666667, 0.6440678 , 0.70833333],
[0.72222222, 0.45833333, 0.74576271, 0.83333333],
[0.66666667, 0.45833333, 0.77966102, 0.95833333],
[0.72222222, 0.45833333, 0.69491525, 0.91666667],
[0.41666667, 0.29166667, 0.69491525, 0.75      ],
[0.69444444, 0.5       , 0.83050847, 0.91666667],
[0.66666667, 0.54166667, 0.79661017, 1.        ],
[0.66666667, 0.41666667, 0.71186441, 0.91666667],
[0.55555556, 0.20833333, 0.6779661 , 0.75      ],
[0.61111111, 0.41666667, 0.71186441, 0.79166667],
[0.52777778, 0.58333333, 0.74576271, 0.91666667],
[0.44444444, 0.41666667, 0.69491525, 0.70833333]])
```

```
scaler = preprocessing. StandardScaler()
# transform the feature
standardized = scaler.fit_transform(features)
standardized
print("Mean {}".format(round (standardized.mean())))
print("Standard Deviation: {}".format(standardized.std()))
```

```
Mean 0
Standard Deviation: 1.0
```

```
# create scaler
robust_scaler = preprocessing. RobustScaler()
#transform feature
robust_scaler.fit_transform (features)
```

```
array([[-0.53846154,  1.        , -0.84285714, -0.73333333],
       [-0.69230769,  0.        , -0.84285714, -0.73333333],
       [-0.84615385,  0.4       , -0.87142857, -0.73333333],
       [-0.92307692,  0.2       , -0.81428571, -0.73333333],
       [-0.61538462,  1.2       , -0.84285714, -0.73333333],
       [-0.30769231,  1.8       , -0.75714286, -0.6       ],
       [-0.92307692,  0.8       , -0.84285714, -0.66666667],
       [-0.61538462,  0.8       , -0.81428571, -0.73333333],
       [-1.07692308, -0.2       , -0.84285714, -0.73333333],
       [-0.69230769,  0.2       , -0.81428571, -0.8       ],
       [-0.30769231,  1.4       , -0.81428571, -0.73333333],
       [-0.76923077,  0.8       , -0.78571429, -0.73333333],
       [-0.76923077,  0.        , -0.84285714, -0.8       ],
       [-1.15384615,  0.        , -0.92857143, -0.8       ],
       [ 0.        ,  2.        , -0.9       , -0.73333333],
       [-0.07692308,  2.8       , -0.81428571, -0.6       ],
       [-0.30769231,  1.8       , -0.87142857, -0.6       ],
       [-0.53846154,  1.        , -0.84285714, -0.66666667],
       [-0.07692308,  1.6       , -0.75714286, -0.66666667],
       [-0.53846154,  1.6       , -0.81428571, -0.66666667],
       [-0.30769231,  0.8       , -0.75714286, -0.73333333],
       [-0.53846154,  1.4       , -0.81428571, -0.6       ],
       [-0.92307692,  1.2       , -0.95714286, -0.73333333],
       [-0.53846154,  0.6       , -0.75714286, -0.53333333],
       [-0.76923077,  0.8       , -0.7       , -0.73333333],
       [-0.61538462,  0.        , -0.78571429, -0.73333333],
       [-0.61538462,  0.8       , -0.78571429, -0.6       ],
       [-0.46153846,  1.        , -0.81428571, -0.73333333],
       [-0.46153846,  0.8       , -0.84285714, -0.73333333],
       [-0.84615385,  0.4       , -0.78571429, -0.73333333],
       [-0.76923077,  0.2       , -0.78571429, -0.73333333],
```

```
[-0.30769231,  0.8       , -0.81428571, -0.6       ],
[-0.46153846,  2.2       , -0.81428571, -0.8       ],
[-0.23076923,  2.4       , -0.84285714, -0.73333333],
[-0.69230769,  0.2       , -0.81428571, -0.73333333],
[-0.61538462,  0.4       , -0.9       , -0.73333333],
[-0.23076923,  1.        , -0.87142857, -0.73333333],
[-0.69230769,  1.2       , -0.84285714, -0.8       ],
[-1.07692308,  0.        , -0.87142857, -0.73333333],
[-0.53846154,  0.8       , -0.81428571, -0.73333333],
[-0.61538462,  1.        , -0.87142857, -0.66666667],
[-1.        , -1.4       , -0.87142857, -0.66666667],
[-1.07692308,  0.4       , -0.87142857, -0.73333333],
[-0.61538462,  1.        , -0.78571429, -0.46666667],
[-0.53846154,  1.6       , -0.7       , -0.6       ],
[-0.76923077,  0.        , -0.84285714, -0.66666667],
[-0.53846154,  1.6       , -0.78571429, -0.73333333],
[-0.92307692,  0.4       , -0.84285714, -0.73333333],
[-0.38461538,  1.4       , -0.81428571, -0.73333333],
[-0.61538462,  0.6       , -0.84285714, -0.73333333],
[ 0.92307692,  0.4       ,  0.1       ,  0.06666667],
[ 0.46153846,  0.4       ,  0.04285714,  0.13333333],
[ 0.84615385,  0.2       ,  0.15714286,  0.13333333],
[-0.23076923, -1.4       , -0.1       ,  0.        ],
[ 0.53846154, -0.4       ,  0.07142857,  0.13333333],
[-0.07692308, -0.4       ,  0.04285714,  0.        ],
[ 0.38461538,  0.6       ,  0.1       ,  0.2       ],
[-0.69230769, -1.2       , -0.3       , -0.2       ],
```

```
import numpy as np
from sklearn.preprocessing import Normalizer
normalizerl1 =Normalizer(norm='l1')
normalizerl2 =Normalizer(norm='l2')
normalizerMax =Normalizer(norm='max')
print("l1 normalization\n",normalizerl1.transform(features))
print("\nl2 normalization\n",normalizerl2.transform(features))
print("\nmax normalization\n",normalizerMax.transform(features))
```

```
[1.         0.5        0.8        0.3       ]
[1.         0.44927536 0.7826087  0.30434783]
[1.         0.46268657 0.8358209  0.35820896]
[1.         0.44927536 0.73913043 0.33333333]
[1.         0.46551724 0.87931034 0.32758621]
[1.         0.47058824 0.86764706 0.33823529]
[1.         0.49253731 0.85074627 0.37313433]
[1.         0.44776119 0.7761194  0.34328358]
[1.         0.3968254  0.79365079 0.3015873 ]
[1.         0.46153846 0.8        0.30769231]
[1.         0.5483871  0.87096774 0.37096774]
[1.         0.50847458 0.86440678 0.30508475]]
```

```
import pandas as pd
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
features,_= make_blobs (n_samples = 150,
                        n_features=3,
                        centers = 3,
                        random_state= 1)
df = pd.DataFrame(features, columns= ["feature_1", "feature_2","feature_3"])
# make k-means clusterer
clusterer = KMeans (3, random_state=0)
# fit clusterer
clusterer.fit(features)
# predict values
df ['group'] = clusterer.predict(features)
df.head()
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change from 10
  warnings.warn(
```

	feature_1	feature_2	feature_3	group
0	-0.941970	4.155785	-10.049242	1
1	-6.475523	-2.792061	-3.001938	2
2	-6.785592	-1.208691	-1.827924	2
3	-0.411260	3.648816	-9.409418	1
4	-4.013822	-8.093843	-7.546288	0

Next steps: Generate code with df View recommended plots

```
import numpy as np
features[~np.isnan(features).any(axis=1)]
```

```
array([[ -0.94197008,   4.15578496, -10.04924243],
       [ -6.4755234 ,  -2.79206127,  -3.00193779],
       [ -6.7855919 ,  -1.20869056,  -1.82792398],
       [ -0.41126012,   3.64881573,  -9.40941834],
       [ -4.01382189,  -8.09384267,  -7.54628777],
       [ -3.79439472,   5.22333705,  -8.81354119],
       [ -7.25507738,  -5.05861909,  -1.34389165],
       [ -6.06498262,  -2.8606084 ,  -0.72361683],
       [ -1.23199783,   4.48049947, -10.45088111],
       [ -6.86332082,  -1.66231803,  -2.37843746],
       [ -0.84702472,   4.6173919 ,  -9.57593335],
       [ -4.87044829,  -3.20920741,  -1.23242237],
       [ -6.12205907,  -4.14707068,  -2.166599  ],
       [ -6.07198971,  -2.23698036,  -2.14545449],
       [ -6.64354615,  -2.02316807,  -1.99049784],
       [ -1.5905901 ,   4.87973126, -10.64806753],
       [ -4.40197918,  -6.59432259,  -9.55180633],
       [ -5.88991027,  -4.25764374,  -0.81167373],
       [ -1.47892914,  -7.39720703,  -7.37608159],
       [ -5.28177321,  -6.89951903,  -8.12568374],
       [ -4.39609607,  -4.52586773,  -1.18619469],
       [ -3.07096007,  -6.32891233,  -9.50459007],
       [ -1.85909282,   4.63932217,  -9.84231924],
       [ -5.32987837,  -3.01353107,  -2.4451363 ],
       [ -1.54634187,   3.16374415,  -8.39822943],
       [ -2.85245239,  -5.95397939,  -8.40109229],
       [ -3.0524769 ,  -5.66239553,  -7.93554555],
       [ -5.51136714,  -2.13552151,  -2.49889127],
       [ -4.79217079,  -7.74341308,  -8.21911732],
       [ -2.43866686,   3.62882716,  -9.37514971],
       [  0.02893715,   5.03230134, -11.60907632],
       [ -1.59946217,   4.86891066,  -9.31287601],
```

```
[ -1.55639691,   3.78482302,  -9.72199446],
[ -6.39400275,  -3.81457605,  -3.13087409],
[ -6.57996897,  -1.26301168,  -2.42684438],
[ -1.31270057,   5.77352257,  -9.32399643],
[ -0.74236389,   4.48674046,  -9.76947373],
[ -1.07763472,   3.99628235,  -7.7008464 ],
[ -5.63202111,  -2.55853869,  -0.6185191 ],
[ -4.53936962,  -4.20390713,  -2.19574027],
[ -4.51388547,  -6.924109  ,  -6.38562715],
[ -5.81979901,  -7.93820265,  -7.85766481],
[ -6.39390511,  -5.29714933,  -2.30126622],
[ -4.54736797,  -6.07666574,  -7.97012404],
[ -3.57365667,  -7.18057694,  -8.29984821],
[ -6.47637244,  -4.03144103,  -2.23675048],
[ -1.05540626,   6.79169568, -10.12159583],
[ -2.69615858,  -6.60231014,  -9.06826133],
[ -1.75343694,   4.23671585, -10.53885713],
[ -6.25378449,  -3.35512243,  -2.43646564],
[ -5.50091176,  -3.66043258,  -1.81714196],
[ -6.25460817,  -5.11952152,  -1.80730775],
[ -1.29372156,   6.01399691, -10.23588987],
[ -7.75883232,  -3.04490815,  -2.70598468],
[ -4.72518456,  -7.32984286,  -7.04594588],
[ -1.50104502,   5.2799081 , -10.10909587],
[ -6.44056643,  -2.38973725,  -3.21985689],
```

```
import pandas as pd
df = pd.DataFrame (features, columns= ["feature_1", "feature_2","feature_3"])
df.dropna()
```

	feature_1	feature_2	feature_3
0	-0.941970	4.155785	-10.049242
1	-6.475523	-2.792061	-3.001938
2	-6.785592	-1.208691	-1.827924
3	-0.411260	3.648816	-9.409418
4	-4.013822	-8.093843	-7.546288
...	...	...	...
145	-4.437912	-6.688998	-7.405180
146	-2.701437	-3.605530	-1.793650
147	-5.138206	-3.407788	-2.079798
148	-2.951123	3.558246	-10.164312
149	-6.666073	-2.469441	-2.828343

150 rows × 3 columns

```
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.datasets import make_blobs

features, _ = make_blobs(n_samples=150, n_features=3, random_state=1)

scaler = StandardScaler()
standardized_features = scaler.fit_transform(features)

true_value = standardized_features[0, 0]
standardized_features[0, 0] = np.nan

mean_imputer = SimpleImputer(strategy="mean")
median_imputer = SimpleImputer(strategy="median")
mode_imputer = SimpleImputer(strategy="most_frequent")

features_mean_imputed = mean_imputer.fit_transform(standardized_features)
features_median_imputed = median_imputer.fit_transform(standardized_features)
features_mode_imputed = mode_imputer.fit_transform(standardized_features)

print("True Value: {}".format(true_value))
print("Mean Imputed Value: {}".format(features_mean_imputed[0, 0]))
print("Median Imputed Value: {}".format(features_median_imputed[0, 0]))
print("Mode Imputed Value: {}".format(features_mode_imputed[0, 0]))
```

```
True Value: 1.4286939379208594
Mean Imputed Value: -0.009588549918932624
Median Imputed Value: -0.016984995847737824
Mode Imputed Value: -2.181974840016922
```