```
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)
```

```
       [ 0.38461538, -0.6       ,  0.15714286,  0.33333333],
       [ 0.69230769,  0.6       ,  0.38571429,  0.53333333],
       [ 1.07692308,  0.4       ,  0.47142857,  0.33333333],
       [ 0.30769231, -0.4       ,  0.12857143,  0.33333333],
       [ 0.23076923,  0.        ,  0.15714286,  0.33333333],
       [ 0.46153846, -0.4       ,  0.35714286,  0.53333333],
       [ 1.07692308,  0.        ,  0.41428571,  0.2       ],
       [ 1.23076923, -0.4       ,  0.5       ,  0.4       ],
       [ 1.61538462,  1.6       ,  0.58571429,  0.46666667],
       [ 0.46153846, -0.4       ,  0.35714286,  0.6       ],
       [ 0.38461538, -0.4       ,  0.21428571,  0.13333333],
       [ 0.23076923, -0.8       ,  0.35714286,  0.06666667],
       [ 1.46153846,  0.        ,  0.5       ,  0.66666667],
       [ 0.38461538,  0.8       ,  0.35714286,  0.73333333],
       [ 0.46153846,  0.2       ,  0.32857143,  0.33333333],
       [ 0.15384615,  0.        ,  0.12857143,  0.33333333],
       [ 0.84615385,  0.2       ,  0.3       ,  0.53333333],
       [ 0.69230769,  0.2       ,  0.35714286,  0.73333333],
       [ 0.84615385,  0.2       ,  0.21428571,  0.66666667],
       [ 0.        , -0.6       ,  0.21428571,  0.4       ],
       [ 0.76923077,  0.4       ,  0.44285714,  0.66666667],
       [ 0.69230769,  0.6       ,  0.38571429,  0.8       ],
       [ 0.69230769,  0.        ,  0.24285714,  0.66666667],
       [ 0.38461538, -1.        ,  0.18571429,  0.4       ],
       [ 0.53846154,  0.        ,  0.24285714,  0.46666667],
       [ 0.30769231,  0.8       ,  0.3       ,  0.66666667],
       [ 0.07692308,  0.        ,  0.21428571,  0.33333333]])
```

```
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))
```

```
l1 normalization
 [[0.5        0.34313725 0.1372549  0.01960784]
 [0.51578947 0.31578947 0.14736842 0.02105263]
 [0.5        0.34042553 0.13829787 0.0212766 ]
 [0.4893617  0.32978723 0.15957447 0.0212766 ]
 [0.49019608 0.35294118 0.1372549  0.01960784]
 [0.47368421 0.34210526 0.14912281 0.03508772]
 [0.4742268  0.35051546 0.1443299  0.03092784]
 [0.4950495  0.33663366 0.14851485 0.01980198]
 [0.49438202 0.3258427  0.15730337 0.02247191]
 [0.51041667 0.32291667 0.15625    0.01041667]
 [0.5        0.34259259 0.13888889 0.01851852]
 [0.48       0.34       0.16       0.02      ]
 [0.51612903 0.32258065 0.15053763 0.01075269]
 [0.50588235 0.35294118 0.12941176 0.01176471]
 [0.51785714 0.35714286 0.10714286 0.01785714]
 [0.475      0.36666667 0.125      0.03333333]
 [0.49090909 0.35454545 0.11818182 0.03636364]
 [0.49514563 0.33980583 0.13592233 0.02912621]
 [0.49565217 0.33043478 0.14782609 0.02608696]
 [0.47663551 0.35514019 0.14018692 0.02803738]
 [0.5046729  0.31775701 0.1588785  0.01869159]
 [0.47663551 0.34579439 0.14018692 0.03738318]
 [0.4893617  0.38297872 0.10638298 0.0212766 ]
 [0.48113208 0.31132075 0.16037736 0.04716981]
 [0.46601942 0.33009709 0.18446602 0.01941748]
 [0.51020408 0.30612245 0.16326531 0.02040816]
 [0.48076923 0.32692308 0.15384615 0.03846154]
 [0.5        0.33653846 0.14423077 0.01923077]
 [0.50980392 0.33333333 0.1372549  0.01960784]
 [0.48453608 0.32989691 0.16494845 0.02061856]
 [0.49484536 0.31958763 0.16494845 0.02061856]
 [0.5046729  0.31775701 0.14018692 0.03738318]
 [0.47706422 0.37614679 0.13761468 0.00917431]
 [0.48672566 0.37168142 0.12389381 0.01769912]
 [0.50515464 0.31958763 0.15463918 0.02061856]
 [0.52083333 0.33333333 0.125      0.02083333]
 [0.52380952 0.33333333 0.12380952 0.01904762]
 [0.49       0.36       0.14       0.01      ]
 [0.49438202 0.33707865 0.14606742 0.02247191]
 [0.5        0.33333333 0.14705882 0.01960784]
 [0.4950495  0.34653465 0.12871287 0.02970297]
 [0.53571429 0.27380952 0.1547619  0.03571429]
 [0.48351648 0.35164835 0.14285714 0.02197802]
 [0.46728972 0.3271028  0.14953271 0.05607477]
 [0.45535714 0.33928571 0.16964286 0.03571429]
 [0.50526316 0.31578947 0.14736842 0.03157895]
 [0.47663551 0.35514019 0.14953271 0.01869159]
 [0.4893617  0.34042553 0.14893617 0.0212766 ]
 [0.4953271  0.34579439 0.14018692 0.01869159]
 [0.50505051 0.33333333 0.14141414 0.02020202]
 [0.42944785 0.19631902 0.28834356 0.08588957]
 [0.41025641 0.20512821 0.28846154 0.09615385]
 [0.42073171 0.18902439 0.29878049 0.09146341]
 [0.41984733 0.17557252 0.30534351 0.09923664]
 [0.42207792 0.18181818 0.2987013  0.0974026 ]
 [0.3986014  0.1958042  0.31468531 0.09090909]
```

```
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: T
  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)]
```

```
       [ -1.32876278,   5.54843239, -11.12730766],
       [ -6.9587239 ,  -4.4310899 ,  -2.2453747 ],
       [ -2.18395001,  -7.06810097,  -8.81968824],
       [ -3.83819314,  -5.74245142,  -7.17006937],
       [ -2.46173688,  -6.96370515,  -7.05498997],
       [ -3.81167336,  -5.32749751,  -7.77745933],
       [ -2.75023479,   3.79650461,  -9.69130012],
       [ -4.13234317,  -7.69799858,  -7.67860094],
       [ -5.92173269,  -3.15965836,  -2.8060866 ],
       [ -2.69759867,   3.39701004, -11.05596907],
       [ -2.82912314,  -7.6512853 ,  -6.69323072],
       [ -4.03694801,  -8.90285781,  -8.35929052],
       [ -1.646431  ,   4.60872892,  -9.54276026],
       [ -3.83154246,  -6.41124193,  -8.24238305],
       [ -2.94595602,  -8.57379788,  -7.74245715],
       [ -5.30208507,  -6.47397943,  -8.11527303],
       [ -6.61705498,  -2.50719387,  -2.97176693],
       [ -2.44649791,  -6.54154824,  -7.73791685],
       [ -8.23740877,  -3.23115466,  -1.25835066],
       [ -4.87028304,  -6.1108658 ,  -8.50460492],
       [ -2.72938266,  -8.4953808 ,  -7.2987852 ],
       [ -4.43791156,  -6.68899775,  -7.40517963],
       [ -2.7014367 ,  -3.60552989,  -1.7936504 ],
       [ -5.13820625,  -3.40778771,  -2.07979765],
       [ -2.95112261,   3.55824595, -10.16431208],
       [ -6.66607259,  -2.46944062,  -2.82834266]])
```

```
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
```

```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler


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

# impute values
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)

# compare true and imputed values
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]))
```