Handling Numerical data

errors :-

overflow errors :- input above the highest range

underflow errors :- input under the lowest value

Sklean Scalers,preprocessing data

standard scaler:- sensitive to outliers

z = (x-u)/s

Z= transformed value

x = sample

u = mean

s = standard deviation

MINMAX Scaler :- doesnt reduce the effect of outliers

MaxAbsScaler

RobustScaler:-

1.with median

2.with interquatile

```
import numpy as np
from sklearn import preprocessing
feature = np.array([
    [-500.5],
    [-100.1],
    [0],
    [100.1],
    [900.9]
])

minmax_scaler = preprocessing.MinMaxScaler(feature_range =(0,1))
scaled_feature = minmax_scaler.fit_transform(feature)
scaled_feature
```

```
array([[0.        ],
       [0.28571429],
       [0.35714286],
       [0.42857143],
       [1.        ]])
```

```
scaler = preprocessing. StandardScaler()
# transform the feature
standardized = scaler.fit_transform(feature)
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 (feature)
```

```
array([[-2.5],
       [-0.5],
       [ 0. ],
       [ 0.5],
       [ 4.5]])
```

```
import numpy as np
from sklearn.preprocessing import Normalizer
features  = np.array([
    [0.5,0.5],
    [1.1,3.2],
    [1.5,20.2],
    [1.63,34.4],
    [10.9,3.3]
])
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        0.5       ]
 [0.25581395 0.74418605]
 [0.06912442 0.93087558]
 [0.04524008 0.95475992]
 [0.76760563 0.23239437]]

l2 normalization
 [[0.70710678 0.70710678]
 [0.32507977 0.9456866 ]
 [0.07405353 0.99725427]
```

```
  [0.04733062 0.99887928]
  [0.95709822 0.28976368]]

max normalization
  [[1.          1.        ]
  [0.34375    1.        ]
  [0.07425743 1.        ]
  [0.04738372 1.        ]
  [1.         0.30275229]]
```

Grouping observation using clustering

```
import pandas as pd
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
features,_= make_blobs (n_samples = 50,
                        n_features=2,
                        centers = 3,
                        random_state= 1)
df = pd.DataFrame(features, columns= ["feature_1", "feature_2"])
# 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	group
0	-9.877554	-3.336145	0
1	-7.287210	-8.353986	2
2	-6.943061	-7.023744	2
3	-7.440167	-8.791959	2
4	-6.641388	-8.075888	2

Next steps: Generate code with df View recommended plots

```
import numpy as np
features = np.array([
    [1.1,11.1],
    [2.2,22.2],
    [3.3,33.3],
    [np.nan,55]
])
features[~np.isnan(features).any(axis=1)]
```

```
array([[ 1.1, 11.1],
       [ 2.2, 22.2],
       [ 3.3, 33.3]])
```

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

	feature_1	feature_2
0	-9.877554	-3.336145
1	-7.287210	-8.353986
2	-6.943061	-7.023744
3	-7.440167	-8.791959
4	-6.641388	-8.075888
5	-0.794152	2.104951
6	-2.760179	5.551214
7	-9.946905	-4.590344
8	-0.525790	3.306599
9	-1.981977	4.022436
10	-5.865964	-7.968072
11	-6.834787	-7.391217
12	-6.749247	-10.175429
13	-10.752110	-2.700480
14	-8.508996	-8.657694
15	-2.330806	4.393825
16	-0.197452	2.346349
17	0.085252	3.645283
18	-10.206607	-3.366725
19	-9.158729	-3.022246
20	-1.340521	4.157119
21	-1.831988	3.528631
22	-9.806797	-1.853093
23	-0.758704	3.722762
24	-11.140231	-4.302691
25	-7.812137	-5.349845
26	-2.351221	4.009736
27	-6.878321	-7.743176
28	-1.782450	3.470720
29	-7.371086	-7.325253

30	-7.735544	-7.775664
31	-11.115023	-3.718933
32	-9.697542	-4.305598
33	-10.189548	-4.840978
34	-2.187732	3.333521
35	-2.346733	3.561284
36	-1.927448	4.936845
37	-10.744871	-2.260894
38	-6.866582	-8.034219
39	-7.512011	-6.928720
40	-6.904845	-7.277059
41	-1.617346	4.989305
42	-0.757969	4.908984
43	-9.484783	-4.251441
44	-7.408736	-8.109631
45	-9.509194	-4.028920
46	-8.337910	-3.211304
47	-9.712125	-3.068207
48	-8.866083	-2.433532
49	-7.684883	-7.455196

Imputing Missing data/values

```
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=1000, n_features=2, random_state=1)

# standardize the features
scaler = StandardScaler()
standardized_features = scaler.fit_transform(features)

# replace the first feature's first value with a missing value
true_value = standardized_features[0, 0]
standardized_features[0, 0] = np.nan
```