KNN Imputation

Dataset may have missing values, this may cause problem for many machine learning Algorithms, Its good practices to identify and replace the missing values. This process is know as missing data imputation or imputing. KNN imputation is more effective to handle and replace the missing values.

It is implemented by the KNNimputer() method which contains the following arguments:

Attribute	Remarks
n_neighbors	number of data points to include closer to the missing value
metric	The distance metric to be used for searching values – {nan_euclidean. callable} by default – nan_euclidean
weights	To determine on what basis should the neighboring values be treated values -{uniform , distance, callable} by default- uniform.

Program to illustrate the KNNImputer class.

```
In [44]: import pandas as pd
         import numpy as np
         dataframe=pd.read_csv("horse-colic.csv", header=None, na_values='?')
```

```
In [12]: print(dataframe.head())
```

```
    0   1        2     3      4     5    6    7    8    9  ...    18    19  \
0  2.0   1   530101  38.5   66.0  28.0  3.0  3.0  NaN  2.0  ...  45.0   8.4
1  1.0   1   534817  39.2   88.0  20.0  NaN  NaN  4.0  1.0  ...  50.0  85.0
2  2.0   1   530334  38.3   40.0  24.0  1.0  1.0  3.0  1.0  ...  33.0   6.7
3  1.0   9  5290409  39.1  164.0  84.0  4.0  1.0  6.0  2.0  ...  48.0   7.2
4  2.0   1   530255  37.3  104.0  35.0  NaN  NaN  6.0  2.0  ...  74.0   7.4

    20   21   22  23     24  25  26  27
0  NaN  NaN  2.0   2  11300   0   0   2
1  2.0  2.0  3.0   2   2208   0   0   2
2  NaN  NaN  1.0   2      0   0   0   1
3  3.0  5.3  2.0   1   2208   0   0   1
4  NaN  NaN  2.0   2   4300   0   0   2

[5 rows x 28 columns]
```

```
In [30]: #summarize the number of rows with missing values for each column

         for i in range(dataframe.shape[1]):
             # count number of rows with missing values
             n_miss = dataframe[[i]].isnull().sum()
             perc = n_miss / dataframe.shape[0] * 100
             print('> %d, Missing: %d (%.1f%%)' % (i, n_miss, perc))
             print('>>%d Missing: %d (%.1f%%)'% (i,n_miss,perc))
```

```
> 0, Missing: 1 (0.3%)
>>0 Missing: 1 (0.3%)
> 1, Missing: 0 (0.0%)
>>1 Missing: 0 (0.0%)
> 2, Missing: 0 (0.0%)
>>2 Missing: 0 (0.0%)
> 3, Missing: 60 (20.0%)
>>3 Missing: 60 (20.0%)
> 4, Missing: 24 (8.0%)
>>4 Missing: 24 (8.0%)
> 5, Missing: 58 (19.3%)
>>5 Missing: 58 (19.3%)
> 6, Missing: 56 (18.7%)
>>6 Missing: 56 (18.7%)
> 7, Missing: 69 (23.0%)
>>7 Missing: 69 (23.0%)
> 8, Missing: 47 (15.7%)
>>8 Missing: 47 (15.7%)
> 9, Missing: 32 (10.7%)
```

```
> 24, Missing: 0 (0.0%)
>>24 Missing: 0 (0.0%)
> 25, Missing: 0 (0.0%)
>>25 Missing: 0 (0.0%)
> 26, Missing: 0 (0.0%)
>>26 Missing: 0 (0.0%)
> 27, Missing: 0 (0.0%)
>>27 Missing: 0 (0.0%)
```

In [31]:
```python
data = dataframe.values
```

In [33]:
```python
from numpy import isnan
from pandas import read_csv
from sklearn.impute import KNNImputer
```

In [34]:
```python
#Dividing the input and output column
ix=[i for i in range(data.shape[1]) if i != 23]
X,y=data[:,ix],data[:,23]
```

In [36]:
```python
X.shape
```

Out[36]:
```
(300, 27)
```

In [37]:
```python
y.shape
```

Out[37]:
```
(300,)
```

In [38]:
```python
# summarize total missing
print('Missing: %d' % sum(isnan(X).flatten()))
```

```
Missing: 1605
```

In [51]:
```python
# define imputer
imputer = KNNImputer()
# fit on the dataset
imputer.fit(X)

# transform the dataset
Xtrans = imputer.transform(X)
# summarize total missing
print('Missing: %d' % sum(isnan(Xtrans).flatten()))
```

```
Missing: 0
```

In [56]:
```python
xtrans=pd.DataFrame(Xtrans)
```


In [56]:
```python
xtrans=pd.DataFrame(Xtrans)
```


In [59]:
```python
xtrans.head(10)
```

Out[59]:

	0	1	2	3	4	5	6	7	8	9	...	17	18	19	20	21	22	23	24	25	26
0	2.0	1.0	530101.0	38.50	66.0	28.0	3.0	3.0	2.2	2.0	...	5.0	45.0	8.40	2.2	3.96	2.0	11300.0	0.0	0.0	2.0
1	1.0	1.0	534817.0	39.20	88.0	20.0	3.0	2.0	4.0	1.0	...	2.0	50.0	85.00	2.0	2.00	3.0	2208.0	0.0	0.0	2.0
2	2.0	1.0	530334.0	38.30	40.0	24.0	1.0	1.0	3.0	1.0	...	1.0	33.0	6.70	2.2	5.18	1.0	0.0	0.0	0.0	1.0
3	1.0	9.0	5290409.0	39.10	164.0	84.0	4.0	1.0	6.0	2.0	...	3.2	48.0	7.20	3.0	5.30	2.0	2208.0	0.0	0.0	1.0
4	2.0	1.0	530255.0	37.30	104.0	35.0	3.0	2.6	6.0	2.0	...	4.2	74.0	7.40	2.4	2.80	2.0	4300.0	0.0	0.0	2.0
5	2.0	1.0	528355.0	38.02	47.2	19.2	2.0	1.0	3.0	1.0	...	3.0	44.4	7.78	1.2	5.22	1.0	0.0	0.0	0.0	2.0
6	1.0	1.0	526802.0	37.90	48.0	16.0	1.0	1.0	1.0	1.0	...	5.0	37.0	7.00	2.8	4.00	1.0	3124.0	0.0	0.0	2.0
7	1.0	1.0	529607.0	38.26	60.0	27.6	3.0	1.8	3.0	1.0	...	4.0	44.0	8.30	2.4	4.90	2.0	2208.0	0.0	0.0	2.0
8	2.0	1.0	530051.0	38.22	80.0	36.0	3.0	4.0	3.0	1.0	...	5.0	38.0	6.20	1.6	4.28	3.0	3205.0	0.0	0.0	2.0
9	2.0	9.0	5299629.0	38.30	90.0	37.2	1.0	2.6	1.0	1.0	...	4.0	40.0	6.20	1.0	2.20	1.0	0.0	0.0	0.0	1.0

10 rows × 27 columns