Numerical dataset processing(steps)

July 19, 2024

steps in in pyton program 1- Import the dependencies(libraries and functions) import numpy as np #for arrays import pandas as pd #for building pd DataFrame from sklearn.preprocessing import StandardScaler #transforming the data values into a common range. from sklearn.model_selection import train_test_split #to split our data in to training and testing data

2- Data Collection and Pre-processing ==to load our data from csv to Pandas DataFrame.

3- Spliting the dataset into features and target. You have to drop the the target column from feature column. When droping the column the axis = 1 and for dropping the row thw axis =0

4 - Data Standardization using Standard Scaler function== transforming the data values into a common range.

from sklearn.preprocessing import StandardScaler #transforming the data values into a common range.

5- Spliting the dataset into Training data and Testing data from sklearn.model_selection import train_test_split #to split our data in to training and testing data

[3]:
```
import numpy as np   #for arrays
import pandas as pd   #for building pd DataFrame
from sklearn.preprocessing import StandardScaler #transforming the data␣
 ↪values into a common range.
from sklearn.model_selection import train_test_split   #to split our data in to␣
 ↪training and testing data
```

[4]:
```
#Data Collection and Pre-processing

diabetes = pd.read_csv('diabetes.csv')
```

[5]:
```
#importing the first five rows
diabetes.head()
```

[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	\
0	6	148	72	35	0	33.6	
1	1	85	66	29	0	26.6	
2	8	183	64	0	0	23.3	
3	1	89	66	23	94	28.1	
4	0	137	40	35	168	43.1	

```
   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[7]:
```
#number of rows and column
diabetes.shape
```

[7]:
```
(768, 9)
```

[8]:
```
#statistics measures of the dataset
diabetes.describe()
```

[8]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
75%     36.600000                  0.626250   41.000000    1.000000
max     67.100000                  2.420000   81.000000    1.000000
```

Spliting the dataset into features and target. You have to drop the the target column from feature column. When droping the column the axis = 1 and for dropping the row thw axis =0

-The first 8 columns are called features -Outcome column is the target

The column of interest is the Outcome

[11]:
```
#Seperating Features and Target

x = diabetes.drop(columns = 'Outcome', axis = 1)
y = diabetes['Outcome']
```

[13]:
```
print(x)
```

```
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0              6      148             72             35        0  33.6
1              1       85             66             29        0  26.6
2              8      183             64              0        0  23.3
3              1       89             66             23       94  28.1
4              0      137             40             35      168  43.1
..           …      …              …              …       …   …
763           10      101             76             48      180  32.9
764            2      122             70             27        0  36.8
765            5      121             72             23      112  26.2
766            1      126             60              0        0  30.1
767            1       93             70             31        0  30.4

     DiabetesPedigreeFunction  Age
0                       0.627   50
1                       0.351   31
2                       0.672   32
3                       0.167   21
4                       2.288   33
..                        …  …
763                     0.171   63
764                     0.340   27
765                     0.245   30
766                     0.349   47
767                     0.315   23

[768 rows x 8 columns]
```

[14]:
```
print(y)
```

```
0      1
1      0
2      1
3      0
4      1
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

0 –> Non - Diabetic 1 –> Diabetic

#Data Standardization using Standard Scaler it is good to standarddized the data before spliting, testing and trainig the data because reads the original range of the dataset. If you split before it can lose some data and beco,es a problem ata that time

[20]:
```
scaler = StandardScaler()
```

[22]:
```
standardized_data = scaler.fit_transform(x)
```

[24]:
```
print(standardized_data)
```

```
[[ 0.63994726  0.84832379  0.14964075 …  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 … -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 … -1.10325546  0.60439732
  -0.10558415]
 …
 [ 0.3429808   0.00330087  0.14964075 … -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 … -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 … -0.20212881 -0.47378505
  -0.87137393]]
```

N/B the dataset x above is in the range of -1 to +1

[25]:
```
x = standardized_data
```

[26]:
```
# N/B x= standarddized data
x is standardized
print(x)
```

```
[[ 0.63994726  0.84832379  0.14964075 …  0.20401277  0.46849198
   1.4259954 ]
 [-0.84488505 -1.12339636 -0.16054575 … -0.68442195 -0.36506078
  -0.19067191]
 [ 1.23388019  1.94372388 -0.26394125 … -1.10325546  0.60439732
  -0.10558415]
 …
 [ 0.3429808   0.00330087  0.14964075 … -0.73518964 -0.68519336
  -0.27575966]
 [-0.84488505  0.1597866  -0.47073225 … -0.24020459 -0.37110101
   1.17073215]
 [-0.84488505 -0.8730192   0.04624525 … -0.20212881 -0.47378505
  -0.87137393]]
```

[29]:
```
print(y)
```

```
0      1
1      0
2      1
3      0
4      1
```

```
      ..
763    0
764    0
765    0
766    1
767    0
Name: Outcome, Length: 768, dtype: int64
```

Spliting the dataset into Training data and Testing data

[32]:
```
x_train, x_test, y_train, y_test = train_test_split(x ,y, test_size = 0.2,␣
↪random_state = 2)
```

[34]:
```
print(x.shape, x_train.shape, x_test.shape)
```

```
(768, 8) (614, 8) (154, 8)
```

[35]:
```
print(y.shape, y_train.shape, y_test.shape)
```

```
(768,) (614,) (154,)
```

[ ]:

[ ]:

[ ]: