Data Analytics Course - Lesson 04

Ths. Vu Duy Khuong

Agenda

- ❑Cleaning and Preparing Dataset - Part 2
  - I. Data Transformation
  - II. Encode Categorical Data

I. Data Transformation

1. Introduction

- In a dataset, the range of values, distributions, units of measure, etc. of the features can vary, depending on the specific problem.

- These differences increase the difficulty of the problem being modeled, or worse, make the model more biased towards features with larger range, larger distributions, etc.

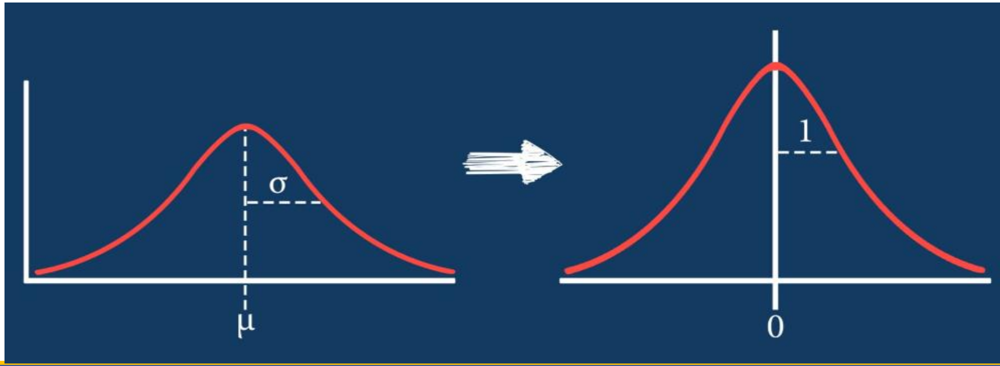
- To avoid this problem, the data needs to be transformed before it can be used to train the model.

I. Data Transformation

1. Introduction

- The two most common techniques for performing transforms on numerical data are ***Normalization*** and ***Standardization***.

- Normalization converts the range of values of all features to 0 to 1.
- Standardization converts the distribution of all features to a normal distribution (which is a distribution with mean = 0, and standard deviation std = 1).



I. Data Transformation

2. Normalization

- The formula for calculating:

$$y = \frac{x - min}{max - min}$$

- x is the initial value of feature.
- y is the value after normalized of the feature.
- min/max is the maximum/minimum value of the feature in the entire data set.

I. Data Transformation

2. Normalization

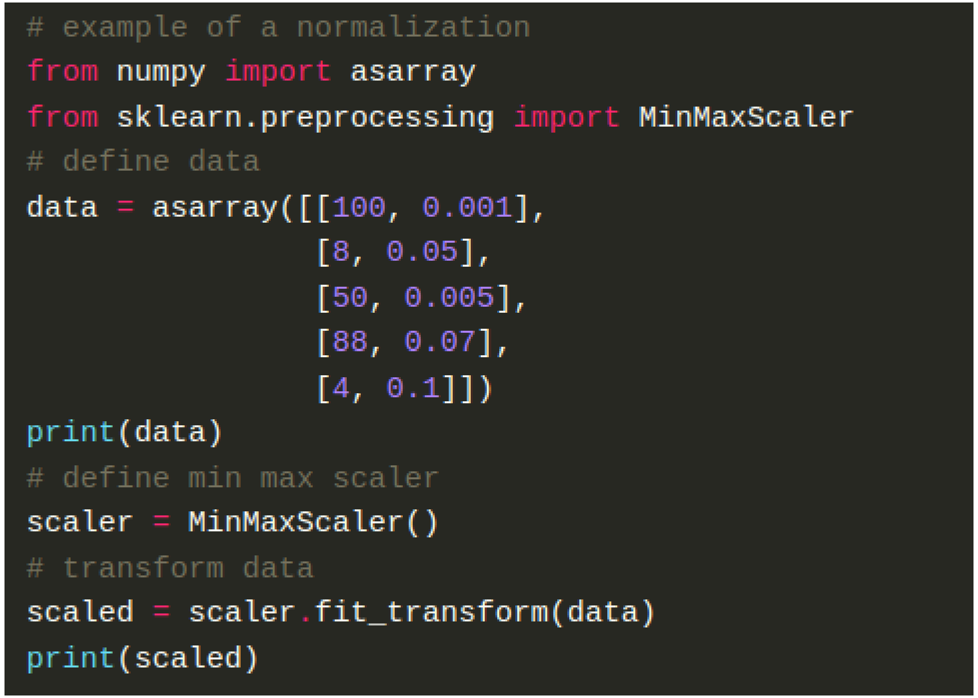
The ***scikit-learn*** library provides the ***MinMaxScaler()*** class to help us do this simply. The steps are as follows:

- Declare an instance of class ***MinMaxScaler()***.
- Fit instance just created on the train set (actually find the ***max/min*** value of each feature in the train set) using the ***fit()*** function.
- Apply the fit instance to the ***train/test*** set, and new data samples later, using the ***transforms()*** function.

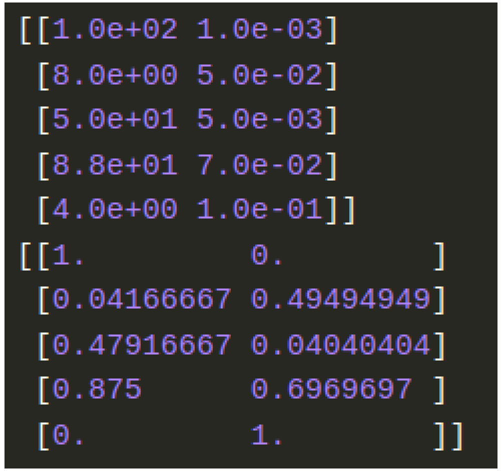
The two functions ***fit()*** and ***transform()*** can be combined into the function ***fit_transform()*** if we only have 1 data set to process (eg, only train set, ...)

I. Data Transformation

2. Normalization



```python
# example of a normalization
from numpy import asarray
from sklearn.preprocessing import MinMaxScaler
# define data
data = asarray([[100, 0.001],
                [8, 0.05],
                [50, 0.005],
                [88, 0.07],
                [4, 0.1]])
print(data)
# define min max scaler
scaler = MinMaxScaler()
# transform data
scaled = scaler.fit_transform(data)
print(scaled)
```



```
[[1.0e+02 1.0e-03]
 [8.0e+00 5.0e-02]
 [5.0e+01 5.0e-03]
 [8.8e+01 7.0e-02]
 [4.0e+00 1.0e-01]]
[[1.         0.        ]
 [0.04166667 0.49494949]
 [0.47916667 0.04040404]
 [0.875      0.6969697 ]
 [0.         1.        ]]
```

I. Data Transformation

2. Normalization

- By default ***MinMaxScaler()*** will return the value of features to the range ***[0,1]***, however, we can specify another desired range via the ***feature_range*** parameter.

- In the opposite direction, we can ***reverse-convert*** the normalized values back to their original values using the ***inverse_transform()*** function of the ***MinMaxScaler()*** class.

- This is useful when we want to find the exact prediction of the model again, because the result returned from the model is the normalized value, which we want to know the value when it has not been normalized.

I. Data Transformation

3. Standardization

The formula for calculating the value of feature when performing standardized is as follows:

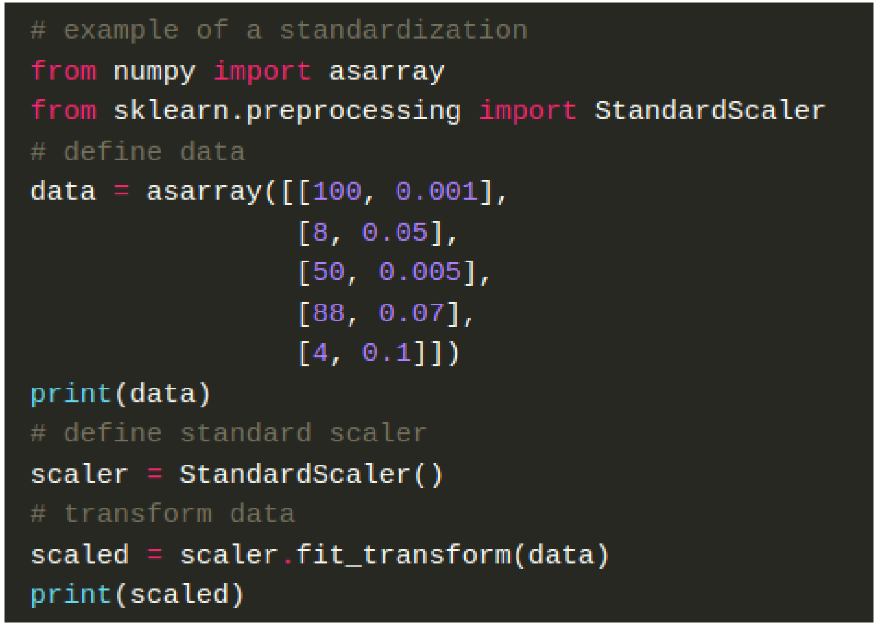
$$y = \frac{x - mean}{std}$$

- x is the initial value of feature.
- y is the value after standardized feature.
- mean is the mean of the feature. $y = \frac{1}{N}\sum_{i=1}^{n} x_i$
- std is the standard deviation value of feature. $y = \sqrt{\frac{\sum_{i=1}^{n}(x_i - mean)^2}{N-1}}$

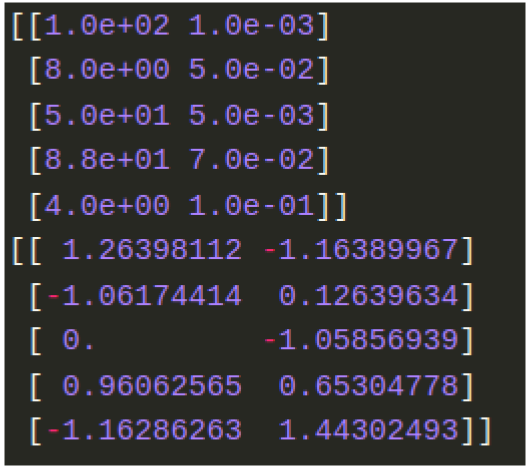
I. Data Transformation

A. Standardization

Scikit-learn provides the StandardScaler() class to perform standardization. Usage is completely similar to MinMaxScaler() class.



```python
# example of a standardization
from numpy import asarray
from sklearn.preprocessing import StandardScaler
# define data
data = asarray([[100, 0.001],
                [8, 0.05],
                [50, 0.005],
                [88, 0.07],
                [4, 0.1]])
print(data)
# define standard scaler
scaler = StandardScaler()
# transform data
scaled = scaler.fit_transform(data)
print(scaled)
```



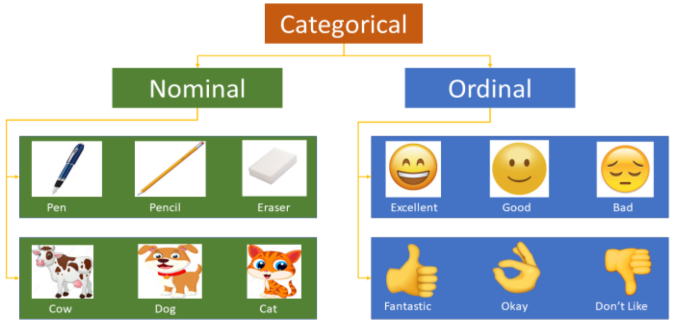
```
[[1.0e+02 1.0e-03]
 [8.0e+00 5.0e-02]
 [5.0e+01 5.0e-03]
 [8.8e+01 7.0e-02]
 [4.0e+00 1.0e-01]]
[[ 1.26398112 -1.16389967]
 [-1.06174414  0.12639634]
 [ 0.         -1.05856939]
 [ 0.96062565  0.65304778]
 [-1.16286263  1.44302493]]
```

II. Encode Categorical Data

1. Introduction

Categorical data types can be divided into 2 types:

- ***Nominal***: Features consisting of a finite number of values that have no ordinal relation to each other. For example, Feature Animals includes values: Dog, Cat, Rabit, Chicken, ...
- ***Ordinal***: Features consist of a finite number of values that are ordinal related. For example: Feature Rank includes values: First, Second, Third, ...



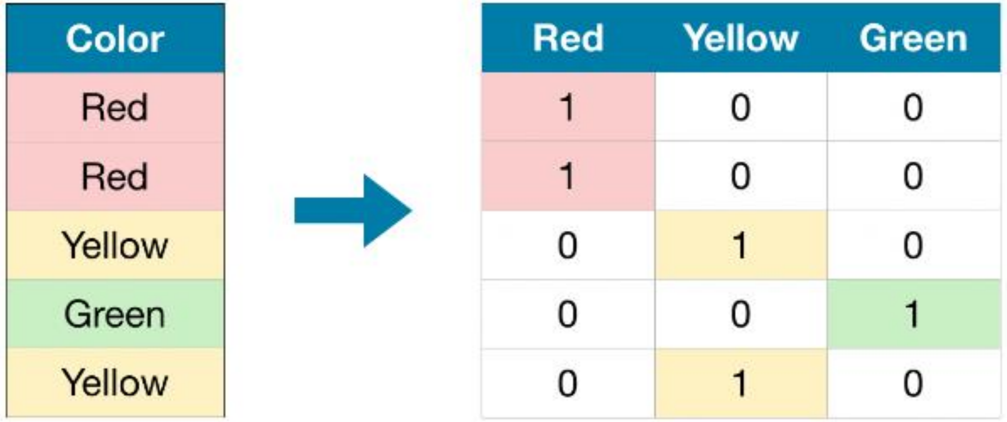
II. Encode Categorical Data

1. Introduction

- Most ML algorithms cannot work directly with categorical data (except Decision Tree). We have to convert the values of features from categorical to numerical before training the models. This conversion process is called Encoding.

- There are three common techniques for encoding categorical data:

- Ordinal Encoding
- One Hot Encoding
- Dummy Variable Encoding



Color
Red
Red
Yellow
Green
Yellow

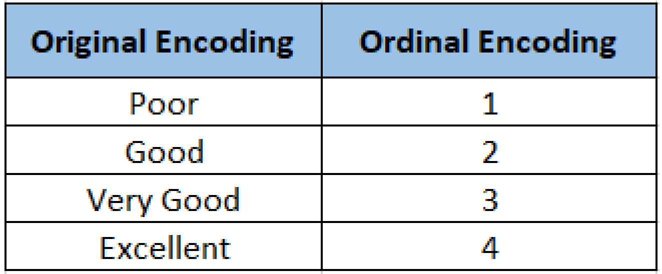
Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

II. Encode Categorical Data

2. Ordinal Encoding

- Ordinal Encoding stipulates that each unique value of a Feature is a natural number, starting from zero.

- For example, exam result have the values: Poor, Good, Very Good, and Excellent, then ***Poor:1, Good:2,Very Good:2, and Excellent:4***. Because natural numbers are inherently ordinal, this approach is only suitable for ordinal categorical data.

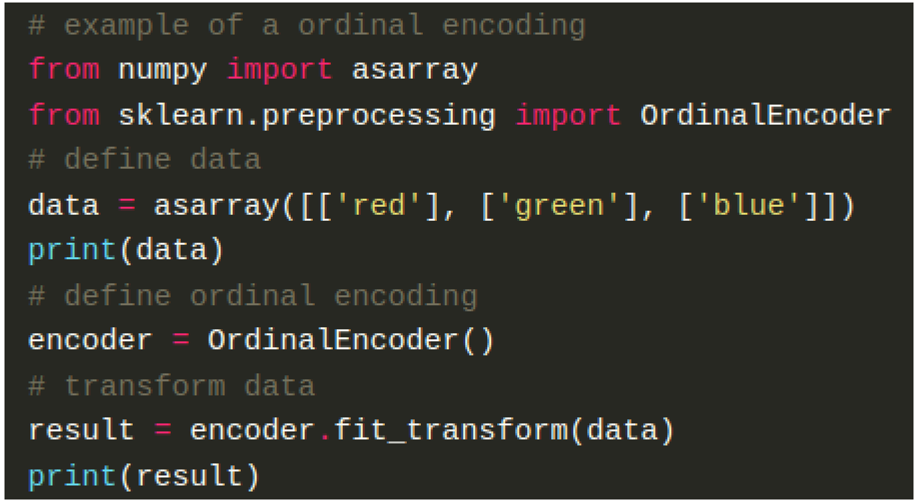


Original Encoding	Ordinal Encoding
Poor	1
Good	2
Very Good	3
Excellent	4

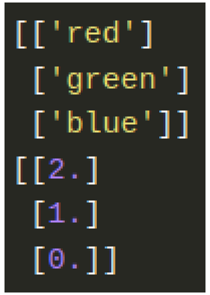
II. Encode Categorical Data

2. Ordinal Encoding

In ***scikit-learn***, Ordinal Encoding is implemented by ***OrdinalEncoder()*** class.



```
# example of a ordinal encoding
from numpy import asarray
from sklearn.preprocessing import OrdinalEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define ordinal encoding
encoder = OrdinalEncoder()
# transform data
result = encoder.fit_transform(data)
print(result)
```



```
[['red']
 ['green']
 ['blue']]
[[2.]
 [1.]
 [0.]]
```

Ordinal Encoding only applies to features, for the target, we use ***LabelEncoder()***. The way these two classes work is completely similar.

II. Encode Categorical Data

3. One-hot Encoding

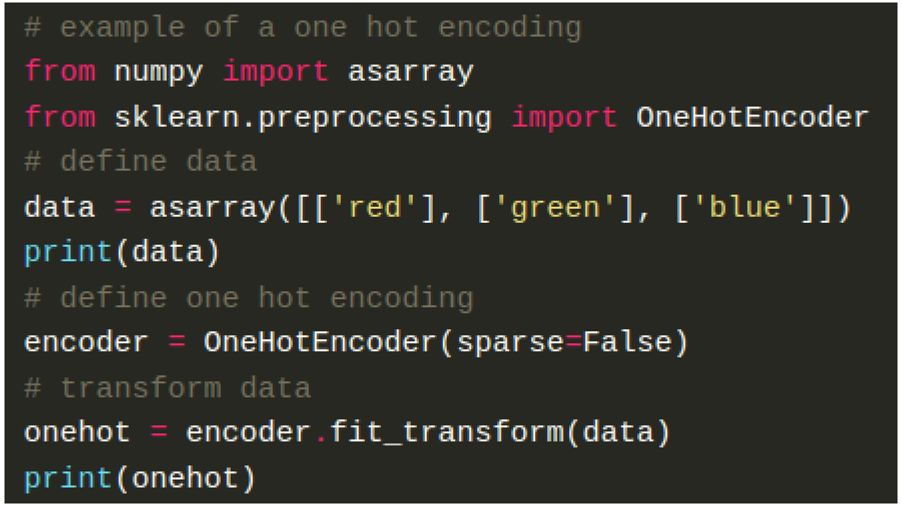
- One Hot Encoding applies to categorical features that are nominal.

- Each feature value is converted to a binary value (including the numbers 0 and 1) where the number of 0s and 1s equals the number of unique values of the feature, the number 1 appearing only at one position in the array represents the unique value of that feature, the other positions in the binary number carry the value 0.

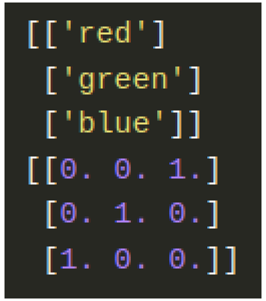
	Cat	Dog	Zebra
	1	0	0
	0	1	0
	0	0	1

II. Encode Categorical Data

3. One-hot Encoding



```
# example of a one hot encoding
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define one hot encoding
encoder = OneHotEncoder(sparse=False)
# transform data
onehot = encoder.fit_transform(data)
print(onehot)
```



```
[['red']
 ['green']
 ['blue']]
[[0. 0. 1.]
 [0. 1. 0.]
 [1. 0. 0.]]
```

II. Encode Categorical Data

4. Dummy Variable Encoding

- One Hot Encoding generates a binary number representing each unique value of the feature. However, it is easy to see that this representation is a bit redundant.

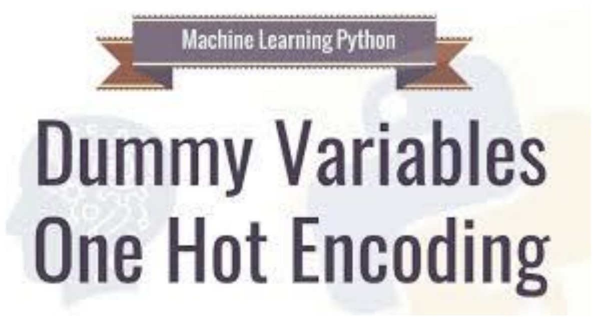
- For example: Suppose our feature has 3 unique values, A, B, C. If we know [1,0,0] is A, [0,1,0] is B, then the remaining values [0, 0.1] is definitely C. Then we might not need a binary number to represent C anymore. In general, if we have N unique values of the feature, then we only need N-1 binary values to represent that feature.

Pandas
get_dummies()

II. Encode Categorical Data

4. Dummy Variable Encoding

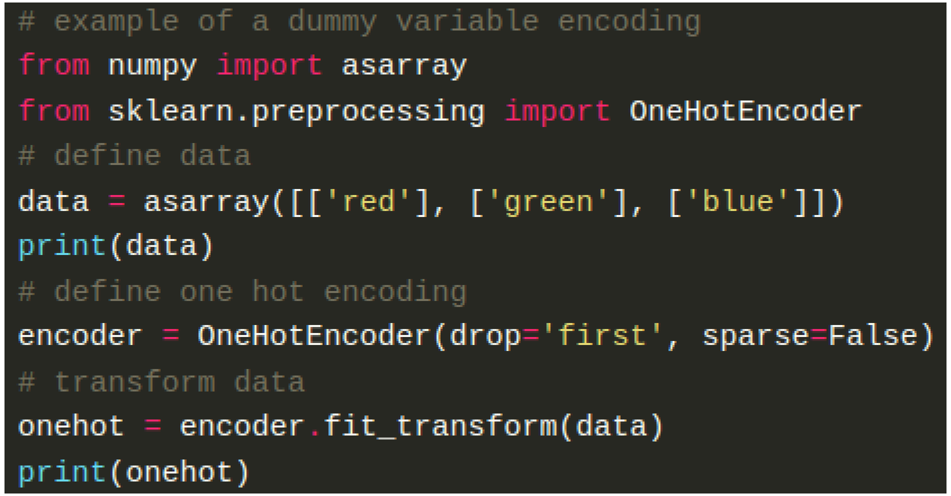
Keep the same idea but change the way of doing it a bit. We still keep N binary values to represent N unique values of the feature, but the length of each binary value is reduced by 1. Specifically: ***A -> [1,0], B -> [0,1], C -> [0,0].***



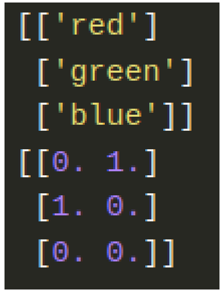
II. Encode Categorical Data

4. Dummy Variable Encoding

To apply ***Dummy Variable Encoding*** in ***scikit-learn***, we still use the ***OneHotEncoder()*** class and set the value of the drop parameter to first.



```
# example of a dummy variable encoding
from numpy import asarray
from sklearn.preprocessing import OneHotEncoder
# define data
data = asarray([['red'], ['green'], ['blue']])
print(data)
# define one hot encoding
encoder = OneHotEncoder(drop='first', sparse=False)
# transform data
onehot = encoder.fit_transform(data)
print(onehot)
```



```
[['red']
 ['green']
 ['blue']]
[[0. 1.]
 [1. 0.]
 [0. 0.]]
```

III. Outlier Data

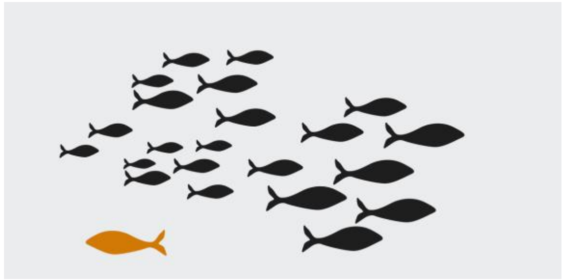
1. Introduction

- Sometimes when working with data, we come across very strange patterns, which are much different from others in the same data set.

- Such differences can be in the data type, value range, data distribution, etc. The number of outliers is not large, and they are called Outlier Data, or "outlier data".

- The existence of Outlier Data can cause noise, making modeling more difficult.

- Outlier Data erasure solution, in most cases, greatly improves the efficiency of ML models.



III. Outlier Data

1. Introduction

- ***Outlier Data*** is born due to one of the following main reasons:

- An error occurred during measurement and data collection.
- Data is damaged (corruption) during storage or conversion.
- The data is affected by random external factors.

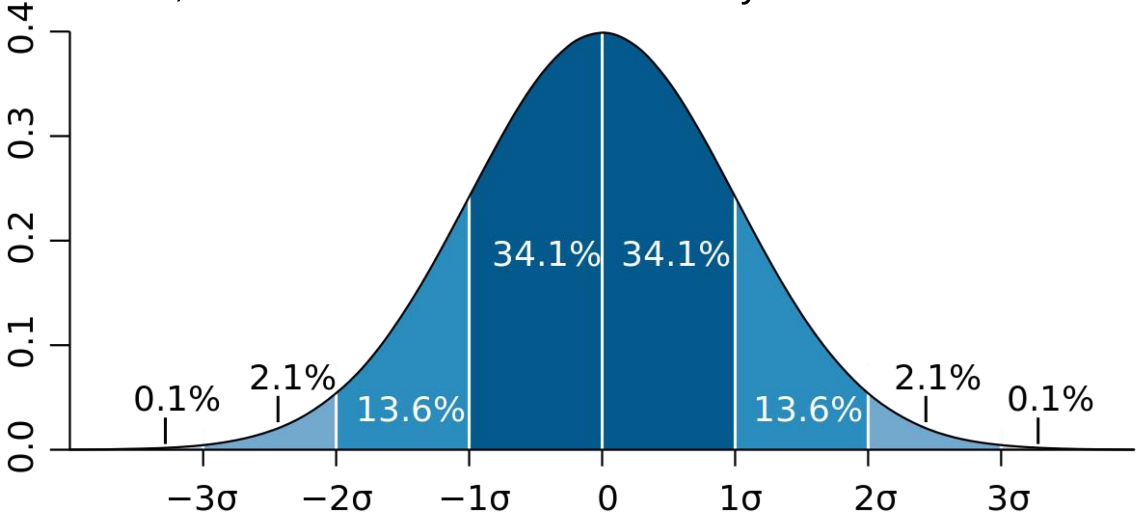
- In practice, there is no exact definition of ***Outlier Data***, because it depends on the specific problem. For one problem, it could be ***Outlier Data***, but for another, it could be a valid value. To make the right decision, it is necessary to consult more experts in that field.

III. Outlier Data

2. Outlier Data Detection

- Method using standard deviation - STD

If we know in advance that the dataset we are working with follows a ***Gaussian*** distribution or is close to ***Gaussian***, then we can use ***STD*** to identify and eliminate ***Outlier Data***.

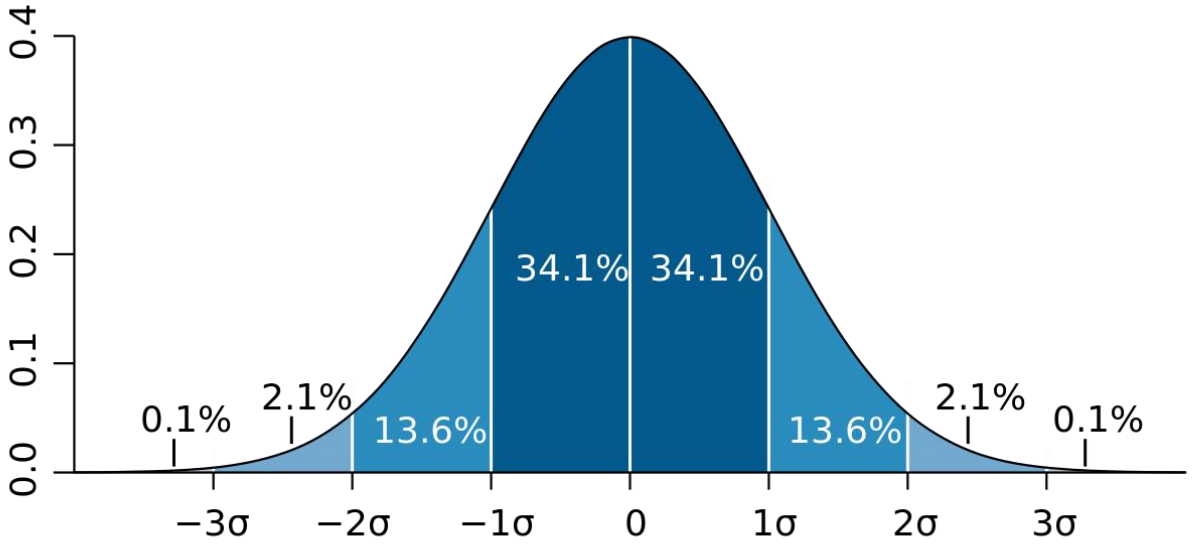


III. Outlier Data

2. Outlier Data Detection

- Method using standard deviation - STD

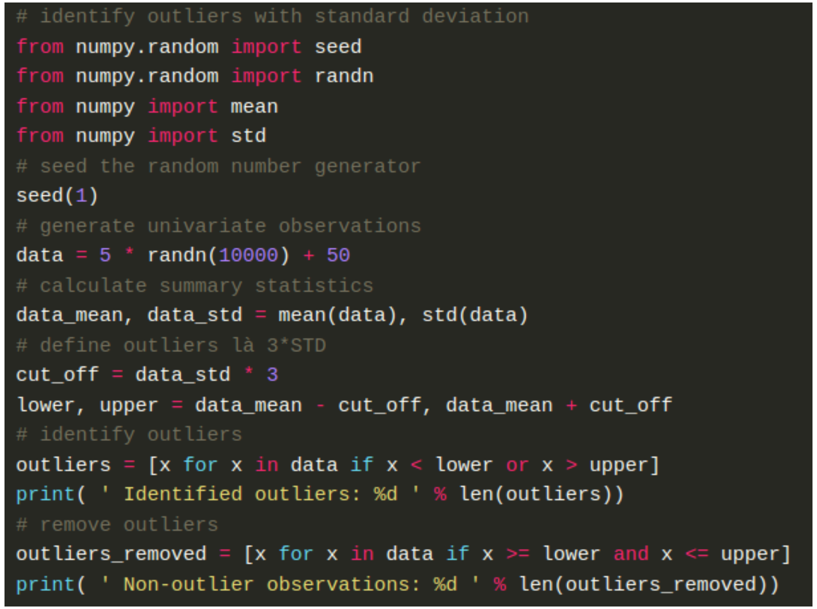
→Defines ***Outlier Data*** as samples that are outside the range of ***STD*** (or ***2STD***, or ***3STD***) from ***Mean***



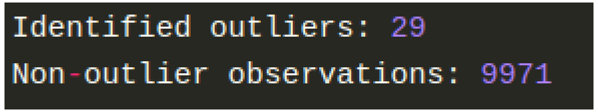
III. Outlier Data

2. Outlier Data Detection

- Method using standard deviation - STD



```python
# identify outliers with standard deviation
from numpy.random import seed
from numpy.random import randn
from numpy import mean
from numpy import std
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate summary statistics
data_mean, data_std = mean(data), std(data)
# define outliers là 3*STD
cut_off = data_std * 3
lower, upper = data_mean - cut_off, data_mean + cut_off
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
print( ' Identified outliers: %d ' % len(outliers))
# remove outliers
outliers_removed = [x for x in data if x >= lower and x <= upper]
print( ' Non-outlier observations: %d ' % len(outliers_removed))
```



```
Identified outliers: 29
Non-outlier observations: 9971
```

III. Outlier Data

2. Outlier Data Detection

- Method using Interquartile Range

Recall a little bit about mean, median, and quartiles.

- Suppose we have the following sequence of numbers: ***6, 5, 8, 7, 12, 13, 15, 14, 2, 200, 1***. The question is to find the mean, median, and quartiles of the sequence of numbers. there.
- The mean is the sum of all the numbers, divided by the number of numbers, where the number of numbers is 11 numbers, so the average will be:

$$Mean = \frac{6+5+8+7+12+13+15+14+2+200+1}{11} = 25.72$$

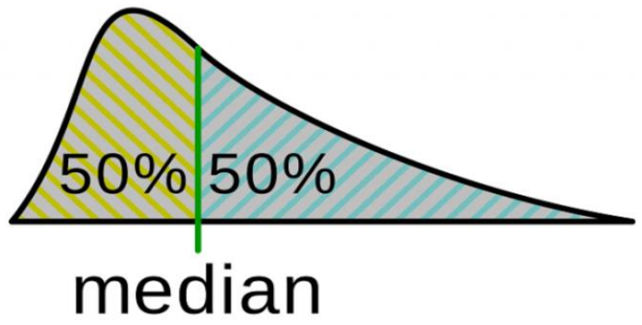
III. Outlier Data

2. Outlier Data Detection

- Method using Interquartile Range

- Median
  - ***Step 1***: Sort the sequence of numbers above in ascending order, we get the results:

    1, 2, 5, 6, 7, 8, 12, 13, 14, 15, 200
  - ***Step 2***: Median is the value that stands in the middle position in an ordered sequence of numbers.

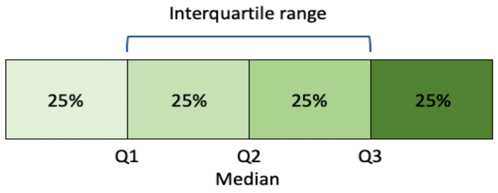


III. Outlier Data

2. Outlier Data Detection

- Method using Interquartile Range

- Interquartile
  - The interquartile is the numerical value that divides a group of numerical observations into ***four parts***, each with equal observations (=***25 %*** of the observations).
  - The quartile has 3 values, the first (***Q1***), the second (***Q2***) and the third (***Q3***) quartile. These three values divide a data set (ordered from smallest to largest) into four equal numbers of observations.



III. Outlier Data

2. Outlier Data Detection

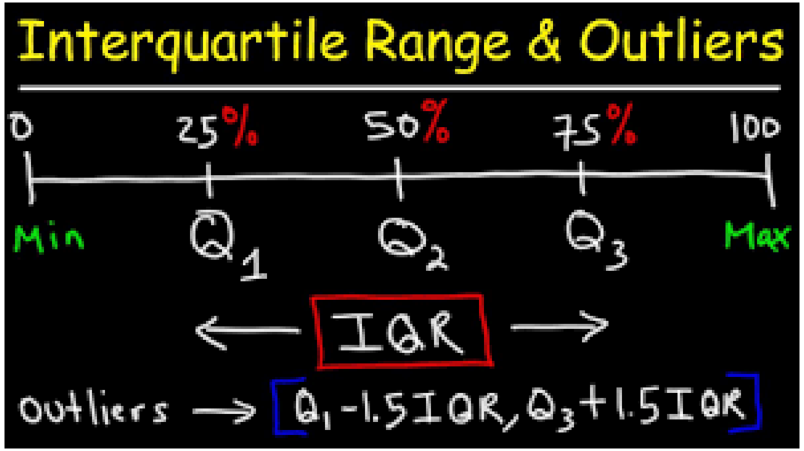
- Method using Interquartile Range

- ***Step 1***: Calculate the error between the ***3rd quartile*** and the ***1st quartile***: I***QR = Q3 - Q1***
- ***Step 2***: Calculate the ***cut_off*** value by multiplying the ***IQR*** by the factor ***k***. The value of ***k*** represents the ***Outlier level*** of the data. Its normal value is ***1.5***: ***cut_off = IQR * k***.
- ***Step 3***: Calculate the ***Outlier upper*** and ***lower*** bounds: ***lower, upper = Q1 - cut_off, Q3 + cut_off***.
- ***Step 4***: Samples whose values are out of range [***lower; upper***] is treated as ***Outlier***.

III. Outlier Data

2. Outlier Data Detection

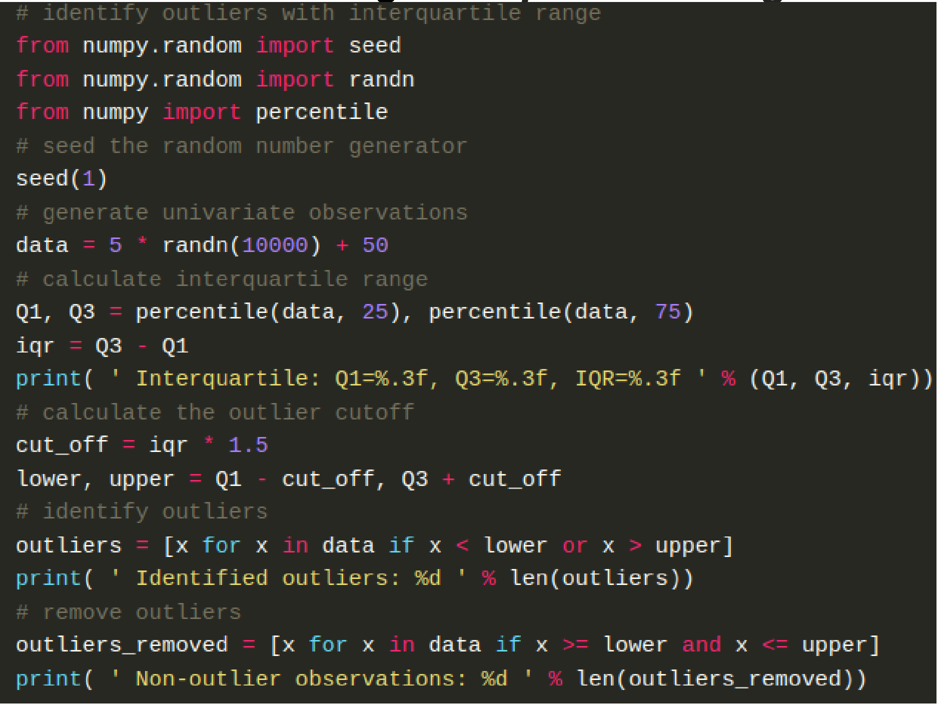
- Method using Interquartile Range


Interquartile Range & Outliers
0
25%
50%
75%
100
Min
Q_1
Q_2
Q_3
Max
IQR
Outliers → $[Q_1 - 1.5\,IQR, Q_3 + 1.5\,IQR]$

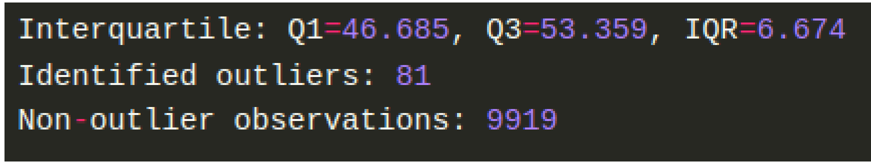
III. Outlier Data

2. Outlier Data Detection

- Method using Interquartile Range



```
# identify outliers with interquartile range
from numpy.random import seed
from numpy.random import randn
from numpy import percentile
# seed the random number generator
seed(1)
# generate univariate observations
data = 5 * randn(10000) + 50
# calculate interquartile range
Q1, Q3 = percentile(data, 25), percentile(data, 75)
iqr = Q3 - Q1
print( ' Interquartile: Q1=%.3f, Q3=%.3f, IQR=%.3f ' % (Q1, Q3, iqr))
# calculate the outlier cutoff
cut_off = iqr * 1.5
lower, upper = Q1 - cut_off, Q3 + cut_off
# identify outliers
outliers = [x for x in data if x < lower or x > upper]
print( ' Identified outliers: %d ' % len(outliers))
# remove outliers
outliers_removed = [x for x in data if x >= lower and x <= upper]
print( ' Non-outlier observations: %d ' % len(outliers_removed))
```



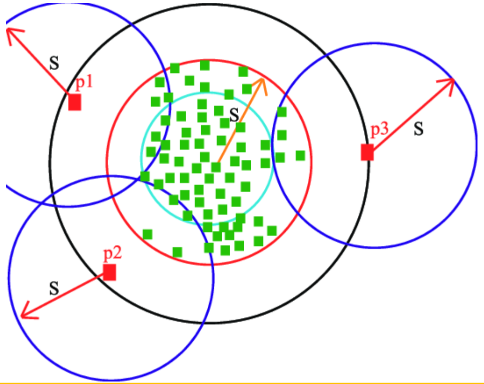
```
Interquartile: Q1=46.685, Q3=53.359, IQR=6.674
Identified outliers: 81
Non-outlier observations: 9919
```

III. Outlier Data

2. Outlier Data Detection

- Method using Local Outlier Factor (LOF)

- ***LOF*** is a technique that exploits the idea of using ***neighbor*** patterns to detect outliers.
- Each sample will be assigned a ***Score*** value representing its isolation level or the likelihood that it could be ***Outlier*** based on the size of its neighborhood.
- The samples with the l***argest Score*** values are more likely to be ***Outliers***.

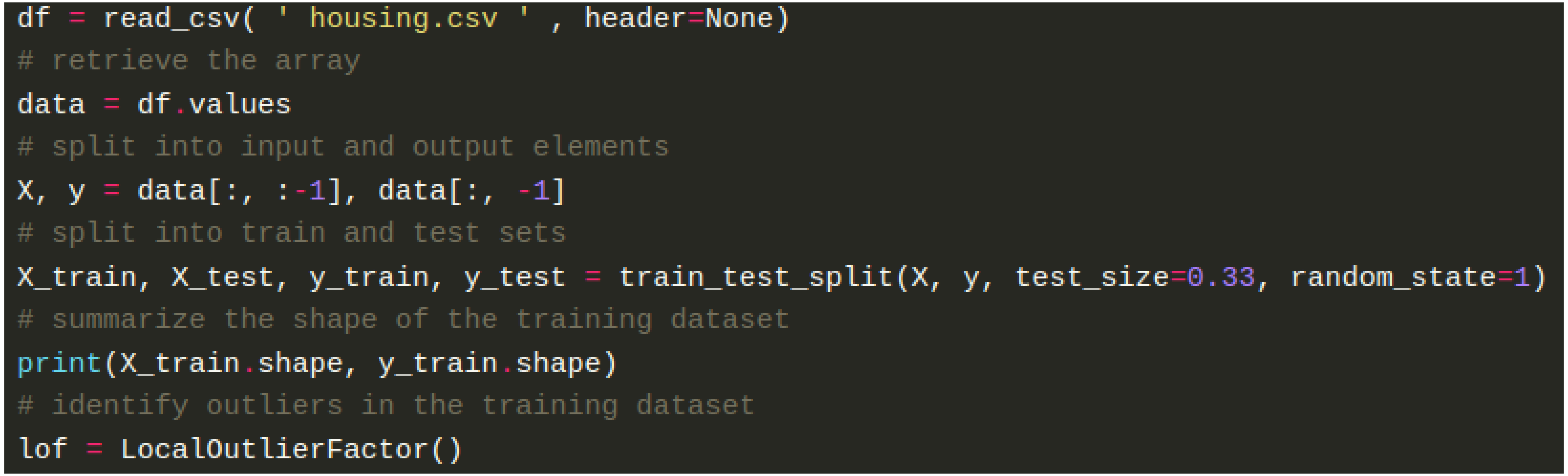


III. Outlier Data

2. Outlier Data Detection

- *Method using Local Outlier Factor (LOF)*

The ***Scikit-learn*** library provides a ***LocalOutlierFactor*** class that simplifies the implementation of this method.



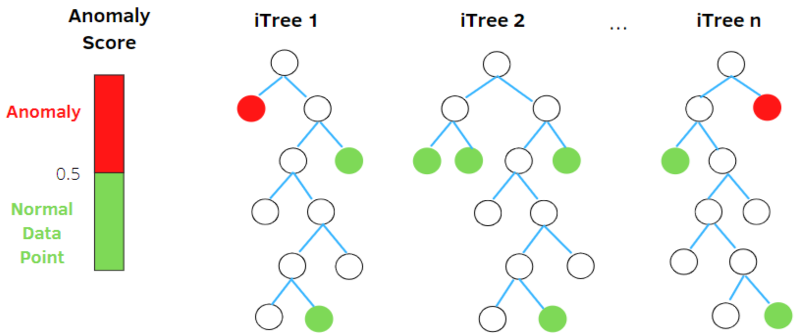
```
df = read_csv( ' housing.csv ' , header=None)
# retrieve the array
data = df.values
# split into input and output elements
X, y = data[:, :-1], data[:, -1]
# split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)
# summarize the shape of the training dataset
print(X_train.shape, y_train.shape)
# identify outliers in the training dataset
lof = LocalOutlierFactor()
```

III. Outlier Data

2. Outlier Data Detection

- Method using Local Outlier Factor (LOF)

- In addition to ***LocalOutlierFactor***, ***Scikit-learn*** also provides another class, ***IsolationForest*** (with a different algorithm, of course) to remove ***Outliers***.
- The two ways to use it are exactly the same.



IV. Reference

Book:

Feature Engineering for Machine Learning, chapter 2, 5

Q & A