# Section4: Sklearn

## SCIKIT LEARN

# Data Scaling

- Data scaling is the process of transforming the values of the features of a dataset till they are within a specific range, e.g. 0 to 1 or -1 to 1. This is to ensure that no single feature dominates the distance calculations in an algorithm, and can help to improve the performance of the algorithm.

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | bedrooms | bathrooms | sqft_living | floors | waterfront | view | condition | grade | sqft_above | sqft_baser | yr_built | yr_renovat | lat | long | sqft_living15 |
| | 3 | 1 | 1180 | 1 | 0 | 0 | 3 | 7 | 1180 | 0 | 1955 | 0 | 47.5112 | -122.257 | 1340 |
| | 3 | 2.25 | 2570 | 2 | 0 | 0 | 3 | 7 | 2170 | 400 | 1951 | 1991 | 47.721 | -122.319 | 1690 |
| | 2 | 1 | 770 | 1 | 0 | 0 | 3 | 6 | 770 | 0 | 1933 | 0 | 47.7379 | -122.233 | 2720 |
| | 4 | 3 | 1960 | 1 | 0 | 0 | 5 | 7 | 1050 | 910 | 1965 | 0 | 47.5208 | -122.393 | 1360 |
| | 3 | 2 | 1680 | 1 | 0 | 0 | 3 | 8 | 1680 | 0 | 1987 | 0 | 47.6168 | -122.045 | 1800 |
| | 4 | 4.5 | 5420 | 1 | 0 | 0 | 3 | 11 | 3890 | 1530 | 2001 | 0 | 47.6561 | -122.005 | 4760 |
| | 3 | 2.25 | 1715 | 2 | 0 | 0 | 3 | 7 | 1715 | 0 | 1995 | 0 | 47.3097 | -122.327 | 2238 |
| | 3 | 1.5 | 1060 | 1 | 0 | 0 | 3 | 7 | 1060 | 0 | 1963 | 0 | 47.4095 | -122.315 | 1650 |
| 0 | 3 | 1 | 1780 | 1 | 0 | 0 | 3 | 7 | 1050 | 730 | 1960 | 0 | 47.5123 | -122.337 | 1780 |
| 1 | 3 | 2.5 | 1890 | 2 | 0 | 0 | 3 | 7 | 1890 | 0 | 2003 | 0 | 47.3684 | -122.031 | 2390 |
| 2 | 3 | 2.5 | 3560 | 1 | 0 | 0 | 3 | 8 | 1860 | 1700 | 1965 | 0 | 47.6007 | -122.145 | 2210 |
| 3 | 2 | 1 | 1160 | 1 | 0 | 0 | 4 | 7 | 860 | 300 | 1942 | 0 | 47.69 | -122.292 | 1330 |
| 4 | 3 | 1 | 1430 | 1.5 | 0 | 0 | 4 | 7 | 1430 | 0 | 1927 | 0 | 47.7558 | -122.229 | 1780 |
| 5 | 3 | 1.75 | 1370 | 1 | 0 | 0 | 4 | 7 | 1370 | 0 | 1977 | 0 | 47.6127 | -122.045 | 1370 |
| 6 | 5 | 2 | 1810 | 1.5 | 0 | 0 | 3 | 7 | 1810 | 0 | 1900 | 0 | 47.67 | -122.394 | 1360 |
| 7 | 4 | 3 | 2950 | 2 | 0 | 3 | 3 | 9 | 1980 | 970 | 1979 | 0 | 47.5714 | -122.375 | 2140 |
| 8 | 3 | 2 | 1890 | 2 | 0 | 0 | 3 | 7 | 1890 | 0 | 1994 | 0 | 47.7277 | -121.962 | 1890 |
| 9 | 4 | 1 | 1600 | 1.5 | 0 | 0 | 4 | 7 | 1600 | 0 | 1916 | 0 | 47.6648 | -122.343 | 1610 |
| 0 | 2 | 1 | 1200 | 1 | 0 | 0 | 4 | 7 | 1200 | 0 | 1921 | 0 | 47.3089 | -122.21 | 1060 |
| 1 | 3 | 1 | 1250 | 1 | 0 | 0 | 4 | 7 | 1250 | 0 | 1969 | 0 | 47.3343 | -122.306 | 1280 |
| 2 | 4 | 1.75 | 1620 | 1 | 0 | 0 | 4 | 7 | 860 | 760 | 1947 | 0 | 47.7025 | -122.341 | 1400 |
| 3 | 3 | 2.75 | 3050 | 1 | 0 | 4 | 3 | 9 | 2330 | 720 | 1968 | 0 | 47.5316 | -122.233 | 4110 |
| 4 | 5 | 2.5 | 2270 | 2 | 0 | 0 | 3 | 8 | 2270 | 0 | 1995 | 0 | 47.3266 | -122.169 | 2240 |

# Data Scaling

- preprocessing.StandardScaler
- preprocessing.MinMaxScaler
- preprocessing.Normalizer
- preprocessing.MaxAbsScaler
- preprocessing.FunctionTransformer
- preprocessing.Binarizer
- preprocessing.PolynomialFeatures

# Standard Scaler (Standardization)

```
from sklearn.preprocessing import StandardScaler
data = [[3652, 1253],
        [21, 7745],
        [-3695, 150],
        [1525, -963]]

scaler = StandardScaler(with_mean=True, with_std=True)
newdata = scaler.fit_transform(data)
print(newdata)

'''The standard score of a sample x is calculated as:
z = (x - u) / s
where u is the mean of the training samples or zero if with_mean=False,
and s is the standard deviation of the training samples or one if with_std=False.'''
```

# MinMaxScaler (Normalization)

```
from sklearn.preprocessing import MinMaxScaler
data=[[-1,2],
      [-0.5,6], # 6 scaled to 0.25
      [0,10],
      [1,18]]

scaler=MinMaxScaler(feature_range=(0,1)) # scaled to range 0 to 1 , max=1, min=0
newdata=scaler.fit_transform(data)
print(newdata)
'''
X_std = (X - X.min) / (X.max- X.min) (Ex: second column, x.min=2 ,x.max=18)
X_scaled = X_std * (max - min) + min
'''
```

# MinMaxScaler (Normalization)

```
from sklearn.preprocessing import MinMaxScaler
data=[[-1,2],
      [-0.5,6], # 6 scaled to 2
      [0,10],
      [1,18]]

scaler=MinMaxScaler(feature_range=(1,5)) # scaled to range 1 to 5 , max=5, min=1
newdata=scaler.fit_transform(data)
print(newdata)
'''
X_std = (X - X.min) / (X.max- X.min) (Ex: second column, x.min=2 ,x.max=18)
X_scaled = X_std * (max - min) + min
'''
```

# Data splitting (train_test_split)

```python
import numpy as np
from sklearn.model_selection import train_test_split
X= np.arange(10).reshape((5, 2))
y= list(range(5))
print(X)
print(y)
#-----------------------------------------------------------------------
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=60,shuffle=False
print(X_train)  # X_train.shape to print the dimension of the X_train data
print("--------------")
print(y_train)
print("--------------")
print(X_test)
print("--------------")
print(y_test)
```

```python
''' test size is a number between 0 and 1 '''

''' random state= ensures that the same randomization is used each time you run the code,
    resulting in the same splits of the data '''

''' shuffle is the Boolean object that determines whether to shuffle
   the dataset before applying the split '''
```

# Data splitting (kfold)

```python
from sklearn.model_selection import KFold
import numpy as np

X = np.array([[1, 2], [3, 4], [5, 6], [7, 8]])
y = np.array([1, 2, 3, 4])

kf=KFold(n_splits=4, random_state=None, shuffle=False)
# set the number of folds to be equal to the number of samples in your dataset,
# This approach is known as leave-one-out cross-validation (LOOCV),
# where each sample in the dataset serves as a separate testing set
# while the remaining samples are used for training.
# you cannot have number of splits n_splits=5 greater than the number of samples: n_samples=4.
```

# Iterate over the splits generated by KFold

```
for train_index, test_index in kf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
    print('X_train \n' , X_train)
    print('X_test \n' , X_test)
    print('y_train \n' ,y_train)
    print('y_test \n' , y_test)

''' train_index: This variable stores the indices of the samples that will be used for training
the model in the current fold. These indices correspond to the rows in your dataset.
test_index: This variable stores the indices of the samples that will be used for testing (validation)
the model in the current fold.
Initially, before any iteration of the loop, these variables don't have specific initial values.
They are assigned values by the kf.split(X) function call within the loop. The split() function
generates pairs of indices corresponding to the training and testing subsets for each fold,
and these pairs are then unpacked into train_index and test_index variables in each iteration of
the loop'''
```

# Tensor Flow and Keras library

- **TensorFlow:** TensorFlow is a versatile open-source machine learning framework developed by Google, offering flexibility and scalability for building, training, and deploying various machine learning models.
- **Keras:** Keras is a high-level neural networks API written in Python, designed for quick and easy experimentation with deep learning models, often used in conjunction with TensorFlow for its simplicity and user-friendly interface.
- **Google Colab**: Google Colab is a cloud-based platform that provides free access to Jupyter notebooks with pre-installed machine learning libraries, including TensorFlow and Keras, allowing collaborative development and execution of machine learning projects directly in the browser.

# MNIST Dataset

**Some key features of the MNIST dataset**:

- **Images**: The dataset contains images of handwritten digits, each measuring 28 pixels by 28 pixels. Therefore, each image is represented as a 28x28 matrix of pixel values.
- **Labels**: Each image in the dataset is associated with a label indicating the digit it represents, ranging from 0 to 9.
- **Training and Test Sets**: The MNIST dataset is typically divided into two subsets: a training set and a test set. The training set is used to train machine learning models, while the test set is used to evaluate the performance of the trained models.

# Load the MNIST dataset and access the training and test sets:

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()

print('X Train Shape is : ', x_train.shape) # X Train Shape is :  (60000, 28, 28)
print('X Train  is : ', x_train[5]) # print the image number 5 in the training set
print('---------------------------------------- ')
print('X Test Shape is : ', x_test.shape) # X Test Shape is :  (10000, 28, 28)
print('X Test  is : ', x_test[5]) # print the image number 5 in the testing set
print('---------------------------------------- ')
print('y Train Shape is : ', y_train.shape) # y Train Shape is :  (60000,)
print('y Train is : ', y_train[5]) # y Train is (label) :  2
print('---------------------------------------- ')
print('y Test Shape is : ', y_test.shape) # y Test Shape is :  (10000,)
print('y Test  is : ', y_test[5]) # y Test  is (label): 1

import matplotlib.pyplot as plt
plt.matshow(x_test[5])
plt.matshow(x_train[5])
plt.show()
```

- The **train_test_split** function from scikit-learn is typically used to split a dataset into training and testing sets.
- However, since the MNIST dataset is already pre-split into training and testing sets, you wouldn't normally use **train_test_split** for this purpose.