

---

빅데이터 분석처리 과정

# 데이터 핸들링 데이터 전처리

---

Numpy, Pandas for Machine Learning  
데이터 전처리 - Data Encoding, Scaling

# 본 수업의 내용

---

- Kaggle 홈페이지에서 데이터를 다운로드하는 방법을 익힌다.
- 머신러닝 데이터를 다루는데 필요한 Numpy library, Pandas Library를 익힌다.
- Numpy
  - ndarray 개요
  - ndarray 데이터 타입
  - ndarray 생성 방법 – arrange, zeros, ones
  - ndarray 다루기 – reshape(), 인덱싱, 슬라이싱, 불리안(조건) 검색 및 인덱싱
  - ndarray 정렬 – sort(), argsort()
- Pandas – 데이터 핸들링
  - Kaggle에서 타이타닉 데이터 다운로드 받기
  - 다운로드 받은 데이터를 기반으로 여러가지 데이터 핸들링 기법 숙지하기
- 데이터 전처리
  - 데이터 Encoding – One-Hot Encoding
  - 데이터 Scaling – 데이터 표준화, 정규화

# Numpy 복습

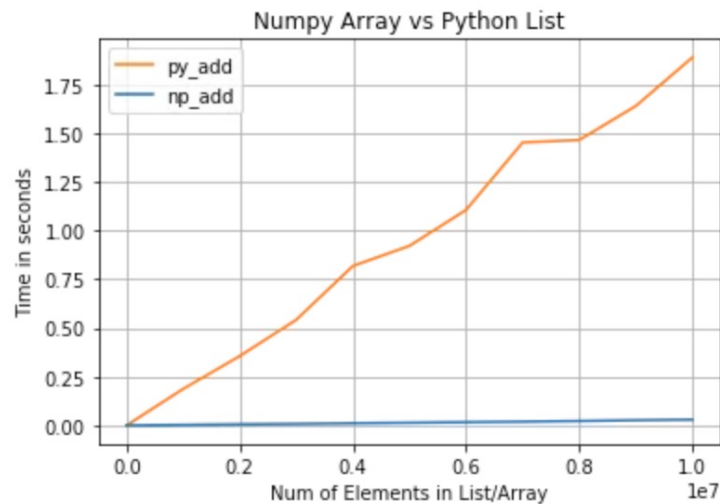
- Numpy for Machine Learning

# Numpy for Machine Learning

- ndarray 개요
- ndarray 데이터 타입
- ndarray 생성 방법 – arrange, zeros, ones
- ndarray 다루기 – reshape(), 인덱싱, 슬라이싱, 불리안(조건) 검색 및 인덱싱
- ndarray 정렬 – sort(), argsort()
- numpy 를 활용한 선형 대수 연산
- numpy로 난수 생성

# Numpy 개요

- 머신러닝의 주요 알고리즘: 선형대수, 통계에 기반
- 파이썬 기반 과학, 공학 패키지: Numpy 의존 패키지가 다수(빠른 계산능력)

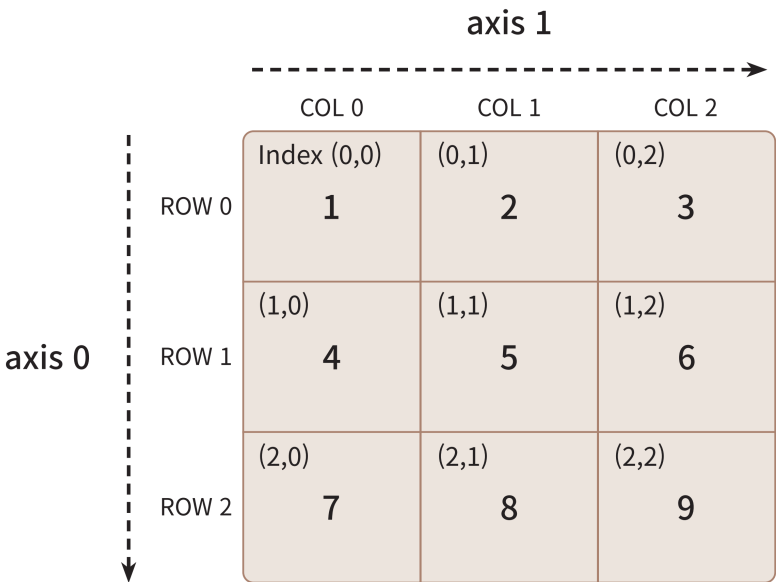
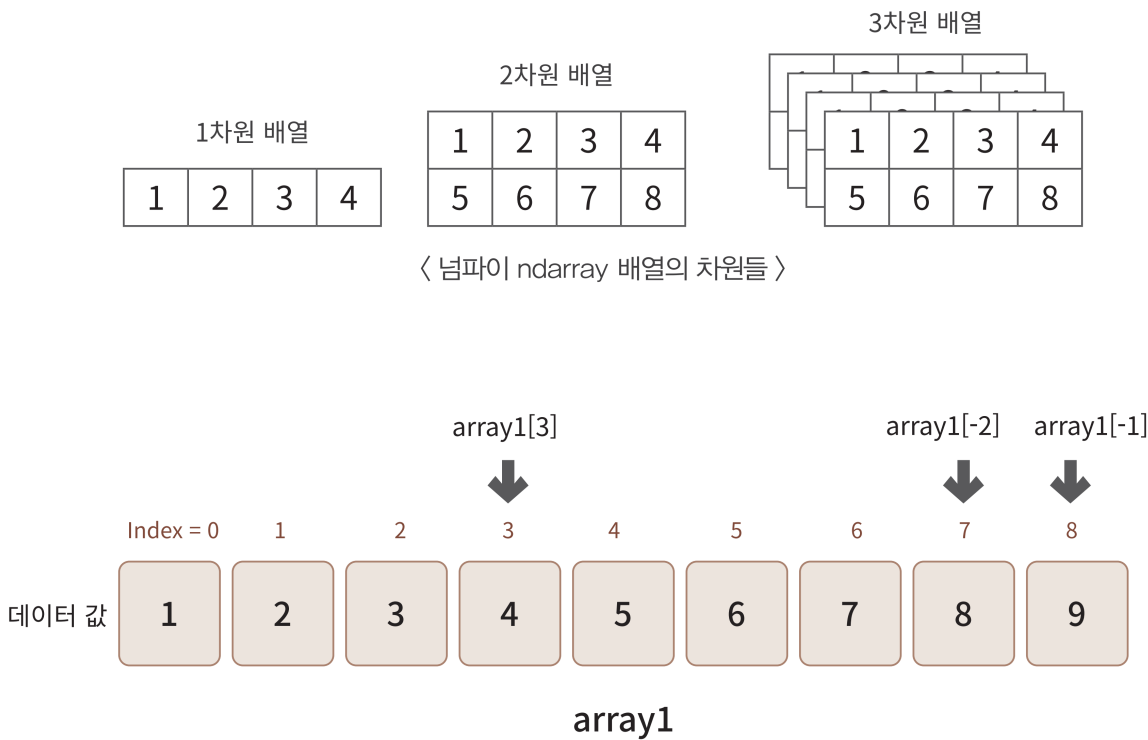


- Numpy(Numerical Python) : 파이썬에서 선형대수 기반의 프로그램을 쉽게 만들 수 있도록 지원
- 빠른 배열 연산 속도: 루프(Loop)를 사용하지 않고 대량 데이터의 배열 연산을 가능하게 함
- 수행 성능 개선을 위해 C/C++ 기반 코드가 많음.

※그림 출처: <https://medium.com/@gough.cory/performance-of-numpy-array-vs-python-list-194c8e283b65>

# Numpy 개요

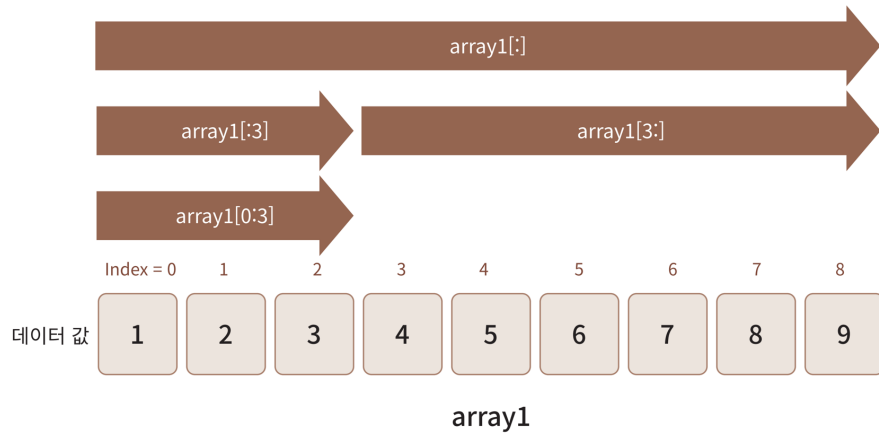
- ndarray 객체
- 넘파이 핵심
  - n차원 배열(동일한 자료형의 array)



※그림 출처: 파이썬 머신러닝 완벽가이드

# Numpy indexing

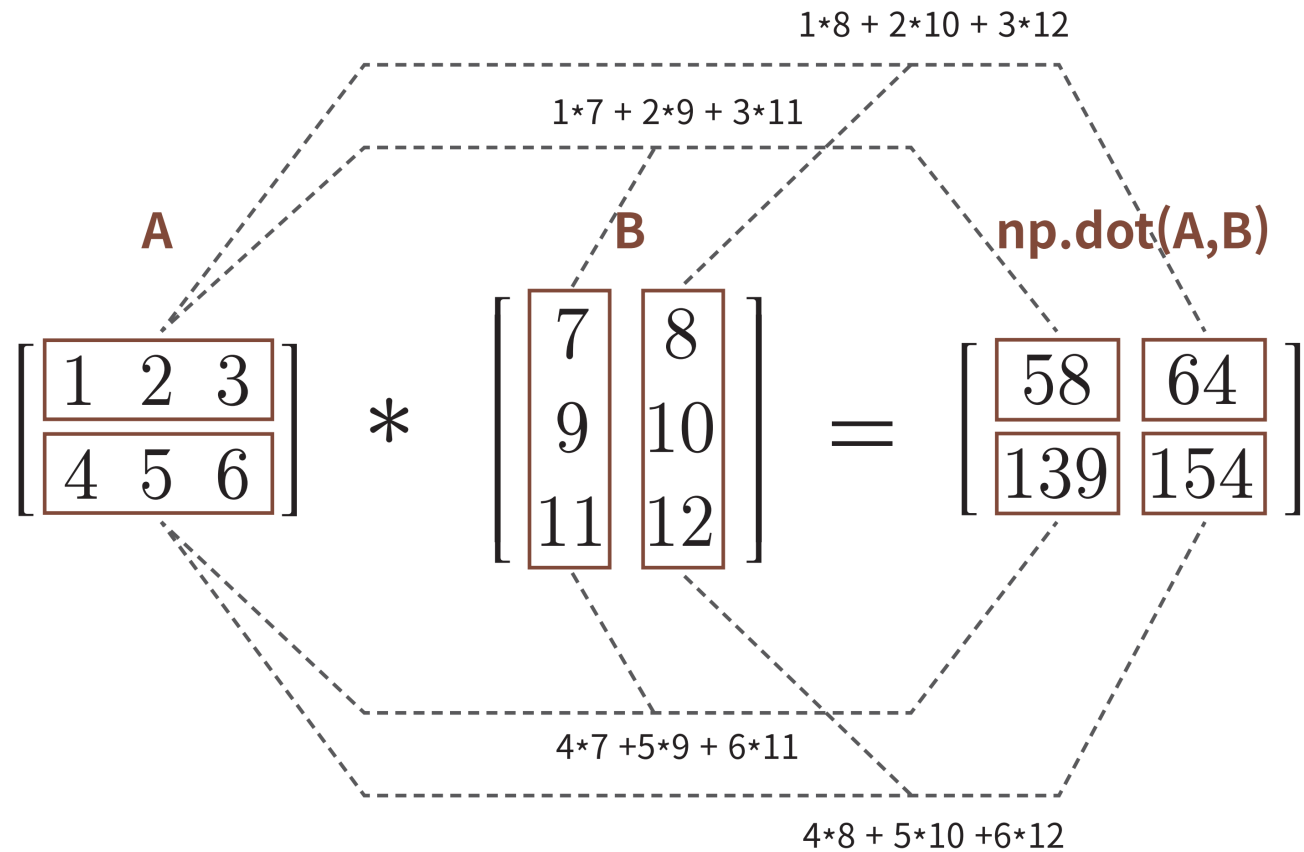
- 파이썬의 리스트와 동일
- Boolean indexing - 조건을 주고 만족하는 값을 추려내는 것
- 다차원 배열의 인덱싱, 슬라이싱



※그림 출처: 파이썬 머신러닝 완벽가이드

# Numpy 선형대수 연산

행렬의 내적(행렬곱)



※그림 출처: 파이썬 머신러닝 완벽가이드



# Numpy 선형대수 연산

---

## 전치행렬

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$$

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

# Numpy로 난수 생성

---

- 시드<sup>seed</sup> 설정하기
  - 의사 난수<sup>Pseudo random number</sup> 이해하기
  - 0.0 ~ 1.0 사이 난수 생성하기
  - 정수 난수 생성하기
  - 정규 분포 난수 생성하기
- 
- 생성한 난수의 평균값, 중앙값 함수 확인하기

# Numpy 실습

# Kaggle 데이터 살펴보기

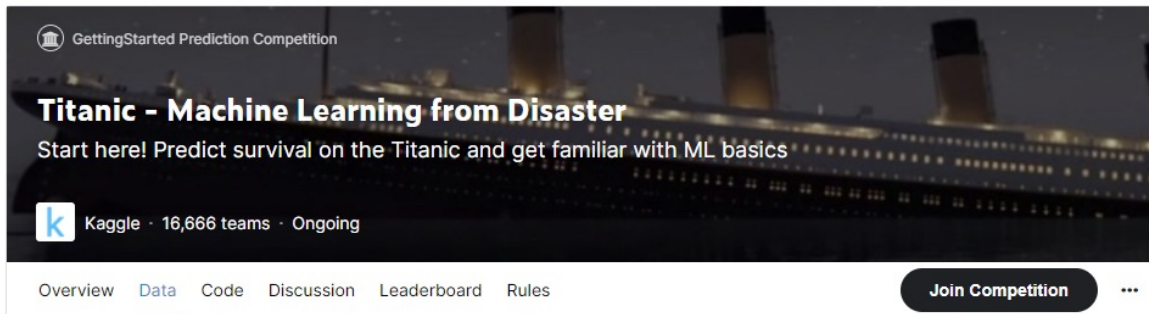
- Kaggle에서 데이터 다운로드 받기
- Pandas DataFrame에서 살펴보기

## Pandas for Data Handling

- DataFrame, Numpy, list, dictionary 변환
- DataFrame 컬럼 데이터셋 생성 , 수정, 삭제
- Data 조회 - 인덱싱, 필터링, 조건 검색
- Kaggle에서 타이타닉 데이터 다운로드 받기
- 다운로드 받은 데이터를 기반으로 여러가지 데이터 핸들링 기법 숙지하기

# Kaggle에서 데이터 다운로드 받기

- [www.kaggle.com](https://www.kaggle.com)
- 로그인 log-in(또는 레지스터 register)
- 경연 참가 규정 준수(I understand and Accept) 클릭
- 타이타닉 탑승자 데이터파일(<https://www.kaggle.com/c/titanic/data>)



GettingStarted Prediction Competition

## Titanic - Machine Learning from Disaster

Start here! Predict survival on the Titanic and get familiar with ML basics

Kaggle · 16,666 teams · Ongoing

Overview Data Code Discussion Leaderboard Rules

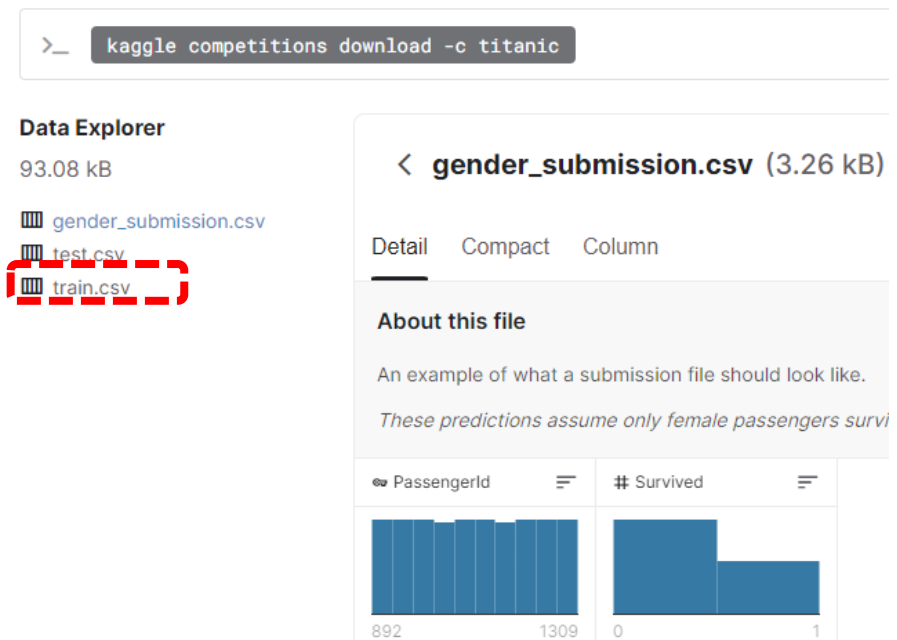
Join Competition

### Data Description

#### Overview

The data has been split into two groups:

- training set (train.csv)
- test set (test.csv)



```
kaggle competitions download -c titanic
```

### Data Explorer

93.08 kB

- gender\_submission.csv
- test.csv
- train.csv

### gender\_submission.csv (3.26 kB)

Detail Compact Column

#### About this file

An example of what a submission file should look like.

*These predictions assume only female passengers survi*

PassengerId	Survived
892	0
1309	1

## 타이타닉 데이터 살펴보기

---

- 파일 이름 변경 : train.csv -> titanic\_train.csv
- 메모장에서 열어보기
- 판다스에서 열어보기

```
import pandas as pd
```

```
titanic_df = pd.read_csv('titanic_train.csv')  
print('titanic 변수 type:', type(titanic_df))  
titanic_df
```

```
titanic 변수 type: <class 'pandas.core.frame.DataFrame'>
```

# DataFrame, 리스트, 딕셔너리, ndarray 변환

---

- 넘파이 ndarray, 리스트, 딕셔너리를 DataFrame으로 변환하기
- DataFrame 열 데이터 생성 및 수정, 삭제
- DataFrame 인덱스 이해하기
- DataFrame 조회 : [], loc[], iloc[], 불리안 인덱싱(조건 검색)
- Data 집계: Groupby(), sort\_values()
- Data 가공: apply, apply lambda



# Pandas 실습

# Machine Learning 데이터 전처리

- 라벨 인코딩(Label Encoding)
- 원핫 인코딩(One Hot Encoding)
- 데이터 스케일링(Data Scaling)

## 데이터 전처리(Data Preprocessing)

- 라벨 인코딩
- 원핫 인코딩
- 데이터 스케일링(표준화, 정규화)

# 데이터 전처리 개요

---

- 데이터 전처리는 ML 알고리즘 만큼 중요.
- Garbage in , Garbage out: 어떤 데이터를 입력으로 가지느냐에 따라 결과도 달라짐
- ML알고리즘을 적용하기 전에 미리 처리해야 할 사항
  - 결손값: Null, Nan은 허용되지 않음.
    - 문자열: 모든 문자열은 숫자형으로 변환
    - 카테고리값, 텍스트형 데이터
- 식별자 삭제
  - 주민번호 등 데이터 행을 식별하는데 사용되는 데이터

## 데이터 인코딩 > 라벨 인코딩 LabelEncoding

### 라벨 인코딩(Label Encoding)

카테고리 피처를 코드형 숫자값으로 변경

원본 데이터	원-핫 인코딩					
상품 분류	상품분류_ TV	상품분류_ 냉장고	상품분류_ 믹서	상품분류_ 선풍기	상품분류_ 전자렌지	상품분류_ 컴퓨터
TV	1	0	0	0	0	0
냉장고	0	1	0	0	0	0
전자렌지	0	0	0	0	1	0
컴퓨터	0	0	0	0	0	1
선풍기	0	0	0	1	0	0
선풍기	0	0	0	1	0	0
믹서	0	0	1	0	0	0
믹서	0	0	1	0	0	0

출처: 파이썬 머신러닝 완벽가이드, p120

```
from sklearn.preprocessing import LabelEncoder

items=['TV','냉장고','전자렌지','컴퓨터','선풍기','선풍기','믹서','믹서']

# LabelEncoder를 객체로 생성한 후, fit() 과 transform() 으로 label 인코딩 수행.
encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)
print('인코딩 변환값:', labels)
```

# 데이터 인코딩 > 원-핫 인코딩One-Hot Encoding

## 원-핫 인코딩(One-Hot Encoding)

피처값의 유형에 따라 새로운 피처를 추가하고 고유값에 해당하는 컬럼만 1, 나머지 컬럼은 0으로 표시하는 방식

원본 데이터		숫자로 인코딩		원-핫 인코딩						
상품 분류	가격	상품 분류	가격	TV	냉장고	믹서	선풍기	전자렌지	컴퓨터	가격
TV	1,000,000	0	1,000,000	1	0	0	0	0	0	1,000,000
냉장고	1,500,000	1	1,500,000	0	1	0	0	0	0	1,500,000
전자렌지	200,000	4	200,000	0	0	0	0	1	0	200,000
컴퓨터	800,000	5	800,000	0	0	0	0	0	1	800,000
선풍기	100,000	3	100,000	0	0	0	1	0	0	100,000
선풍기	100,000	3	100,000	0	0	0	1	0	0	100,000
믹서	50,000	2	50,000	0	0	1	0	0	0	50,000
믹서	50,000	2	50,000	0	0	1	0	0	0	50,000

출처: 파이썬 머신러닝 완벽가이드, p121

## 데이터 인코딩 > 원-핫 인코딩 One-Hot Encoding

### 사이킷런: OneHotEncoder를 이용하는 방식

```
from sklearn.preprocessing import OneHotEncoder
import numpy as np

items=['TV', '냉장고', '전자렌지', '컴퓨터', '선종기', '선종기', '믹서', '믹서']

# 먼저 숫자값으로 변환을 위해 LabelEncoder로 변환합니다.
encoder = LabelEncoder()
encoder.fit(items)
labels = encoder.transform(items)
# 2차원 데이터로 변환합니다.
labels = labels.reshape(-1,1)

# 원-핫 인코딩을 적용합니다.
oh_encoder = OneHotEncoder()
oh_encoder.fit(labels)
oh_labels = oh_encoder.transform(labels)
print('원-핫 인코딩 데이터')
print(oh_labels.toarray())
print('원-핫 인코딩 데이터 차원')
print(oh_labels.shape)
```

### OneHot Encoding 주의점

1. 모든 문자열값을 숫자로 전환
2. 입력값은 2차원 데이터 형태

원-핫 인코딩 데이터

```
[[1. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 0. 1.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0. 0.]
 [0. 0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0. 0.]]
```

원-핫 인코딩 데이터 차원  
(8, 6)

## 데이터 인코딩 > 원-핫 인코딩 One-Hot Encoding

판다스 : get\_dummies()를 이용한 원-핫 인코딩

```
import pandas as pd

df = pd.DataFrame({'item': ['TV', '냉장고', '전자렌지', '컴퓨터', '선풍기', '선풍기', '믹서', '믹서'] })
pd.get_dummies(df)
```

	item_TV	item_냉장고	item_믹서	item_선풍기	item_전자렌지	item_컴퓨터
0	1	0	0	0	0	0
1	0	1	0	0	0	0
2	0	0	0	0	1	0
3	0	0	0	0	0	1
4	0	0	0	1	0	0
5	0	0	0	1	0	0
6	0	0	1	0	0	0
7	0	0	1	0	0	0



# 데이터 스케일링Scaling

---

## 피쳐 스케일링Feature Scaling

서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업, 실제 특성값의 크기와 상관없이 동일한 조건으로 비교

## 표준화(Standardization)

평균 0, 분산 1인 가우시안 정규 분포를 가진 값으로 변환, 각 특성값이 0에서 표준편차의 몇 배만큼 떨어져 있는가를 표시

$$x_{i\_new} = \frac{x_i - \text{mean}(x)}{\text{stdev}(x)}$$

## 정규화(Normalization)

서로 다른 피쳐의 크기를 통일하기 위해 크기를 변환, 동일한 크기 단위로 비교(최소 0 ~ 최대 1의 범위로 변환)

$$x_{i\_new} = \frac{x_i - \min(x)}{\max(x) - \min(x)}$$

## 데이터 스케일링 Scaling > StandardScaler()

### 사이킷런 피쳐 스케일링 Feature Scaling

서로 다른 변수의 값 범위를 일정한 수준으로 맞추는 작업

### StandardScaler()

평균 0, 분산 1인 가우시안 정규 분포를 가진 값으로 변환

데이터가 가우시안 분포를 가지고 있다고 가정하고 구현된 SVM, 선형 회귀, 로지스틱 회귀 등의 알고리즘에서 중요.

```
from sklearn.datasets import load_iris
import pandas as pd
# 붓꽃 데이터 셋을 로딩하고 DataFrame으로 변환합니다.
iris = load_iris()
iris_data = iris.data
iris_df = pd.DataFrame(data=iris_data, columns=iris.feature_names)

print('feature 들의 평균 값')
print(iris_df.mean())
print('feature 들의 분산 값')
print(iris_df.var())
```

```
feature 들의 평균 값
sepal length (cm)    5.843333
sepal width (cm)     3.057333
petal length (cm)    3.758000
petal width (cm)     1.199333
dtype: float64
```

```
feature 들의 분산 값
sepal length (cm)    0.685694
sepal width (cm)     0.189979
petal length (cm)    3.116278
petal width (cm)     0.581006
dtype: float64
```

## 데이터 스케일링 Scaling > MinMaxScaler()

### MinMaxScaler()

데이터 값을 0과 1 사이의 범위 값으로 변환

음수값이 있는 경우 -1 ~ 1 사이의 값으로 변환

데이터 분포가 가우시안 정규분포가 아닌 경우 MiMax Scaling을 고려

```
from sklearn.preprocessing import MinMaxScaler

# MinMaxScaler 객체 생성
scaler = MinMaxScaler()
# MinMaxScaler 로 데이터 셋 변환. fit() 과 transform() 호출.
scaler.fit(iris_df)
iris_scaled = scaler.transform(iris_df)

# transform()시 scale 변환된 데이터 셋이 numpy ndarray로 반환되어 이를 DataFrame으로 변환
iris_df_scaled = pd.DataFrame(data=iris_scaled, columns=iris.feature_names)
print('feature들의 최소 값')
print(iris_df_scaled.min())
print('feature들의 최대 값')
print(iris_df_scaled.max())
```

feature들의 최소 값

sepal length (cm)	0.0
sepal width (cm)	0.0
petal length (cm)	0.0
petal width (cm)	0.0

dtype: float64

feature들의 최대 값

sepal length (cm)	1.0
sepal width (cm)	1.0
petal length (cm)	1.0
petal width (cm)	1.0

dtype: float64

## 데이터 스케일링Scaling > 유의 사항

---

scaling에 사용하는 사이킷런 fit(), transform(), fit\_transform()메소드 유의 사항

fit()

데이터 변환을 위한 기준 정보 설정(예를 들어 데이터 셋의 최댓값/최솟값 설정) 적용

transform()

설정된 정보를 이용해 데이터를 변환

fit\_transform()

fit()과 transform()을 한번에 적용 및 수행

학습 데이터 세트와 테스트 데이터 세트에 fit(), transform() 적용 시 유의사항

1. 가능하다면 전체 데이터의 스케일링 변환을 적용한 뒤 학습, 테스트 데이터 세트로 분리
2. 1항이 안되는 경우 테스트 데이터 변환 시에 fit()이나 fit\_transform()을 적용하지 않고 학습 데이터로 이미 fit()된 Scaler객체를 사용해 transform()으로 변환

## 데이터 스케일링 Scaling > 유의 사항

### 학습 데이터 세트와 테스트 데이터 세트에 fit(), transform() 적용 시 유의사항

- Scaler() 객체를 이용해 학습 데이터 세트로 fit(), transform()을 적용하면 테스트 데이터 세트에는 다시 fit()을 적용하지 않고, 학습 데이터 세트로 fit()을 수행한 결과를 이용해 transform() 변환을 적용해야 한다.
- 즉, 학습 데이터로 fit()이 적용된 스케일링 기준 정보를 그대로 테스트 데이터에 적용해야 하며, 그렇지 않고 테스트 데이터로 다시 새로운 스케일링 기준을 만들지 말아야 한다.

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np

train_array = np.arange(0, 11).reshape(-1, 1)
test_array = np.arange(0, 6).reshape(-1, 1)

# 최소값 0, 최대값 1로 변환하는 MinMaxScaler 객체 생성
scaler = MinMaxScaler()
#train_set
scaler.fit(train_array)
train_scaled = scaler.transform(train_array)

print('원본 train_array 데이터:', train_array.reshape(-1))
print('Scale된 train_array 데이터:', train_scaled.reshape(-1))
```

원본 train\_array 데이터: [ 0 1 2 3 4 5 6 7 8 9 10]  
Scale된 train\_array 데이터: [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]

## 데이터 스케일링Scaling > 유의 사항

학습 데이터 세트와 테스트 데이터 세트에 fit(), transform() 적용 시 유의사항

```
#test_set
scaler.fit(test_array)
test_scaled = scaler.transform(test_array)

# train_array 변환 출력
print('원본 test_array 데이터:', test_array.reshape(-1))
print('Scale된 test_array 데이터:', test_scaled.reshape(-1))
```

원본 test\_array 데이터: [0 1 2 3 4 5]  
Scale된 test\_array 데이터: [0. 0.2 0.4 0.6 0.8 1. ]

```
from sklearn.preprocessing import MinMaxScaler
import numpy as np

train_array = np.arange(0, 11).reshape(-1, 1)
test_array = np.arange(0, 6).reshape(-1, 1)

# 최소값 0, 최대값 1로 변환하는 MinMaxScaler 객체 생성
scaler = MinMaxScaler()
#train_set
scaler.fit(train_array)
train_scaled = scaler.transform(train_array)

print('원본 train_array 데이터:', train_array.reshape(-1))
print('Scale된 train_array 데이터:', train_scaled.reshape(-1))

test_scaled = scaler.transform(test_array)

# train_array 변환 출력
print('원본 test_array 데이터:', test_array.reshape(-1))
print('Scale된 test_array 데이터:', test_scaled.reshape(-1))
```

원본 train\_array 데이터: [ 0 1 2 3 4 5 6 7 8 9 10]  
Scale된 train\_array 데이터: [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]  
원본 test\_array 데이터: [0 1 2 3 4 5]  
Scale된 test\_array 데이터: [0. 0.1 0.2 0.3 0.4 0.5]

# 데이터 전처리 실습

(타이타닉 데이터 세트)

---

# **titanic 데이터세트 전처리 실습**



