머신러닝을 위한 파이썬 핵심정리

NumPy 소개

- Numerical Python
- 파이썬의 내장 타입인 리스트보다 데이터의 저장 및 처리에 있어 효율적인 NumPy 배열을 제공
- 선형 대수와 관련된 기능을 제공
- 파이썬을 기반으로 한 데이터 과학 도구의 핵심 패키지

※ 데이터 사이언스 영역의 대부분의 도구(Pandas, Scipy, scikit-learn 패키지 등)가 Numpy 기반

NumPy의 주요 기능

NumPy패키지와배열(ndarray) 객체

```
1  import numpy as np
```

Numpy를 import할 때
일반적으로 사용하는 별칭

```
1  np.array([1, 4, 2, 5, 3])
```

Array : 동질의 데이터를 다룰 수 있는 구조

```
array([1, 4, 2, 5, 3])
```

```
1  np.array([1, 2, 3, 4], dtype=np.float)
```

데이터 타입

```
array([1., 2., 3., 4.])
```

```
1  np.array([range(i, i + 3) for i in [1, 4, 7]])
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

2차원 구조의
array

NumPy의 주요 기능

```
1  np.zeros(10)
```

1차원으로 만들고 0으로 채움

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.])
```

```
1  np.ones((3, 5))
```

2차원으로 만들고 1으로 채움

```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

```
1  np.full((2, 3), 5)
```

초기값을 임의로 설정

```
array([[5, 5, 5],
       [5, 5, 5]])
```

NumPy의 주요 기능

```
1  np.arange(0, 10, 2)
```

```
array([0, 2, 4, 6, 8])
```

```
1  np.linspace(0, 100, 5, dtype=int)
```

균등하게 interval을 구성

```
array([  0,  25,  50,  75, 100])
```

NumPy의 주요 기능

```
1  np.random.random((3, 3))
```

0~1사이 난수 배열 생성

```
array([[0.56235288, 0.71738895, 0.96925213],
       [0.75709755, 0.78407142, 0.2614919 ],
       [0.54033566, 0.17707708, 0.93600132]])
```

```
1  np.random.randint(0, 10, (3, 3))
```

설정한 구간(0~9까지) 내의 난수 배열 생성

```
array([[7, 9, 1],
       [5, 9, 1],
       [0, 6, 9]])
```

```
1  np.random.normal(0, 1, (3, 3))
```

설정한 평균, 표준편차 정규분포 확률에 의한 난수 배열 생성

```
array([[-1.27670118, -0.85089473, -0.12022143],
       [ 0.27042133,  0.01918626,  0.83638446],
       [ 0.54611631,  0.89703639, -0.97728234]])
```

NumPy의 주요 기능

NumPy 배열 객체의 주요 속성

seed값을 주면 재현성 문제 해결(동일한 seed로 초기화 하면 동일한 순서의 난수가 생성됨)

```
1  np.random.seed(0)
2
3  arr1 = np.random.randint(10, size=6)
4  arr2 = np.random.randint(10, size=(2, 3))
5
6  print("arr1:₩n%s" % arr1)
7  print("ndim: %d, shape: %s, size: %d, dtype: %s₩n" %
8        (arr1.ndim, arr1.shape, arr1.size, arr1.dtype))
9  print("arr2:₩n%s" % arr2)
10 print("ndim: %d, shape: %s, size: %d, dtype: %s" %
11       (arr2.ndim, arr2.shape, arr2.size, arr2.dtype))
```

Ndim : 차원정보
Shape : 행/열 원소 구조 정보
Size : 원소의 개수
Dtype : 데이터 타입

```
arr1:
[5 0 3 3 7 9]
ndim: 1, shape: (6,), size: 6, dtype: int32

arr2:
[[3 5 2]
 [4 7 6]]
ndim: 2, shape: (2, 3), size: 6, dtype: int32
```

NumPy의 주요 기능

NumPy 배열 객체의 인덱싱

단일 원소에 접근하는 기법

```
1  arr1
```

```
array([5, 0, 3, 3, 7, 9])
```

```
1  arr1[0], arr1[5]
```

```
(5, 9)
```

```
1  arr1[-6], arr1[-1]
```

맨 마지막 데이터

```
(5, 9)
```

```
1  arr2
```

```
array([[3, 5, 2],
       [4, 7, 6]])
```

```
1  arr2[0]
```

```
array([3, 5, 2])
```

NumPy의 주요 기능

```
1  arr2[0, 0], arr2[0, 2]
```

[ ]을 사용해 행/열 인덱스 정보 기술

```
(3, 2)
```

```
1  arr2[-1, -3], arr2[-1, -1]
```

```
(4, 6)
```

```
1  arr2[0, 0] = 9
2  arr2
```

```
array([[9, 5, 2],
       [4, 7, 6]])
```

NumPy의 주요 기능

NumPy 배열 객체의 슬라이싱

부분집합 추출
[start:end:step] 형태

```
1  arr1 = np.arange(10)
2  arr1
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
1  arr1[0:5:1]
```

```
array([0, 1, 2, 3, 4])
```

```
1  arr1[:5:1]
```

```
array([0, 1, 2, 3, 4])
```

```
1  arr1[:5:], arr1[:5]
```

```
(array([0, 1, 2, 3, 4]), array([0, 1, 2, 3, 4]))
```

비어있는 슬라이싱 연산자를 활용할 경우
전체집합을 부분집합으로 반환

```
1  arr1[:], arr1[::]
```

```
(array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]), array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9]))
```

NumPy의 주요 기능

```
1  arr1[2:9:2], arr1[2::2]
```

```
(array([2, 4, 6, 8]), array([2, 4, 6, 8]))
```

```
1  arr1[::-1]
```

step이 음수인 경우 거꾸로 출력

```
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

```
1  arr1[-1:-11:-1]
```

```
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

```
1  arr1[5::-1]
```

```
array([5, 4, 3, 2, 1, 0])
```

NumPy의 주요 기능

```
1  arr2 = np.arange(12).reshape(-1, 4)
2  arr2
```

행/열 변환

-1 : 정해지지 않음을 의미
(열을 4개로 맞춰서 3행4열로 변환)

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
1  arr2[:3, :4]
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
1  arr2[:, :]
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

NumPy의 주요 기능

```
1 arr2[:2, :3]
```

```
array([[0, 1, 2],
       [4, 5, 6]])
```

```
1 arr2[:2, 2::-1]
```

```
array([[2, 1, 0],
       [6, 5, 4]])
```

```
1 arr2[1:, -1]
```

```
array([ 7, 11])
```

```
1 arr2[-1, :], arr2[-1]
```

```
(array([ 8,  9, 10, 11]), array([ 8,  9, 10, 11]))
```

NumPy의 주요 기능

NumPy 배열 객체의 연결

```
1  list1 = [1, 2, 3]
2  list2 = [4, 5, 6]
```

```
1  np.concatenate([list1, list2])
```

두개의 list를 [ ] 안에 담아서 전달해야 함

```
array([1, 2, 3, 4, 5, 6])
```

```
1  arr1 = np.concatenate([list1, list2], axis=0)
2  arr1
```

```
array([1, 2, 3, 4, 5, 6])
```

NumPy의 주요 기능

```
1  arr2 = arr1.reshape(-1, 3)
2  arr2
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
1  np.concatenate([arr2, arr2], axis=0)
```

배열의 축(방향)

```
array([[1, 2, 3],
       [4, 5, 6],
       [1, 2, 3],
       [4, 5, 6]])
```

```
1  np.concatenate([arr2, arr2], axis=1)
```

```
array([[1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6]])
```

NumPy의 주요 기능

```
1  np.vstack([arr2, arr2])
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [1, 2, 3],
       [4, 5, 6]])
```

```
1  np.hstack([arr2, arr2])
```

```
array([[1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6]])
```

NumPy의 주요 기능

NumPy배열 객체의 내장 함수 사용

Python의 loop는 속도가 느리기 때문에 numpy 내장함수(vectorize function)를 사용하는 것을 권장

```
1  np.random.seed(0)
2
3  arr2 = np.random.randint(1, 10, (3, 4))
4  arr2
```

```
array([[6, 1, 4, 4],
       [8, 4, 6, 3],
       [5, 8, 7, 9]])
```

```
1  np.sum(arr2), arr2.sum()
```

np.sum(array), array.sum() : array의 합계

```
(65, 65)
```

```
1  np.sum(arr2, axis=0), arr2.sum(axis=0)
```

축 방향으로 합계 계산

```
(array([19, 13, 17, 16]), array([19, 13, 17, 16]))
```

```
1  np.sum(arr2, axis=1), arr2.sum(axis=1)
```

```
(array([15, 21, 29]), array([15, 21, 29]))
```

NumPy의 주요 기능

```
1  np.min(arr2, axis=0), np.max(arr2, axis=0)
```

```
(array([5, 1, 4, 3]), array([8, 8, 7, 9]))
```

np.min(array), array.min() : 최소값
np.max(array), array.max() : 최대값

```
1  arr2.min(axis=0), arr2.max(axis=0)
```

```
(array([5, 1, 4, 3]), array([8, 8, 7, 9]))
```

```
1  np.min(arr2, axis=1), np.max(arr2, axis=1)
```

```
(array([1, 3, 5]), array([6, 8, 9]))
```

```
1  arr2.min(axis=1), arr2.max(axis=1)
```

```
(array([1, 3, 5]), array([6, 8, 9]))
```

NumPy의 주요 기능

NumPy 배열 객체와 브로드캐스팅

행열구조를 동일하게 맞춰주는 것

```
1  np.random.seed(0)
2
3  X = np.random.random((10, 3))
4  X
```

```
array([[0.5488135 , 0.71518937, 0.60276338],
       [0.54488318, 0.4236548 , 0.64589411],
       [0.43758721, 0.891773  , 0.96366276],
       [0.38344152, 0.79172504, 0.52889492],
       [0.56804456, 0.92559664, 0.07103606],
       [0.0871293 , 0.0202184 , 0.83261985],
       [0.77815675, 0.87001215, 0.97861834],
       [0.79915856, 0.46147936, 0.78052918],
       [0.11827443, 0.63992102, 0.14335329],
       [0.94466892, 0.52184832, 0.41466194]])
```

```
1  Xmean = X.mean(axis=0)
2  Xmean
```

```
array([0.52101579, 0.62614181, 0.59620338])
```

NumPy의 주요 기능

X : 10x3 배열
Xmean : 1x3 배열

서로 다른 크기의 배열 간 연산을 위해 브로드캐스팅으로 구조를 맞춤

```
1  Xcentered = X - Xmean
2  Xcentered
```

```
array([[ 0.02779771,  0.08904756,  0.00655999],
       [ 0.02386739, -0.20248701,  0.04969073],
       [-0.08342858,  0.26563119,  0.36745938],
       [-0.13757427,  0.16558323, -0.06730846],
       [ 0.04702877,  0.29945483, -0.52516732],
       [-0.43388649, -0.60592341,  0.23641646],
       [ 0.25714096,  0.24387034,  0.38241496],
       [ 0.27814277, -0.16466245,  0.18432579],
       [-0.40274137,  0.01377921, -0.45285009],
       [ 0.42365312, -0.10429349, -0.18154144]])
```

각 원소에 해당하는 열의 평균을 뺀 결과

NumPy의 주요 기능

NumPy 배열 객체의 부울 배열과 마스킹 연산

```
1  np.random.seed(0)
2
3  X = np.random.randint(1, 10, size=(3, 4))
4  X
```

```
array([[6, 1, 4, 4],
       [8, 4, 6, 3],
       [5, 8, 7, 9]])
```

```
1  (X > 5) & (X < 8)
```

```
array([[ True, False, False, False],
       [False, False,  True, False],
       [False, False,  True, False]])
```

X는 5보다 크고 8보다 작은 조건에 대한 부울 (bool) 배열 생성

```
1  np.sum((X > 5) & (X < 8))
```

```
3
```

X는 5보다 크고 8보다 작은 조건을 만족하는 원소 (bool 배열이 true인 원소)에 대해서만 sum을 계산

```
1  np.sum((X > 5) | (X < 8))
```

```
12
```

NumPy의 주요 기능

```
1  np.sum((X > 5) & (X < 8), axis=0)
```

```
array([1, 0, 2, 0])
```

```
1  np.sum((X > 5) & (X < 8), axis=1)
```

```
array([1, 1, 1])
```

```
1  X[(X > 5) & (X < 8)]
```

조건에 맞는 원소들만 추출

```
array([6, 6, 7])
```

NumPy의 주요 기능

NumPy 배열 객체와 팬시 인덱싱

인덱스 배열을 만족하는 부분집합을 추출

```
1  X = np.arange(12).reshape((3, 4))
2  X
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
1  row = np.array([0, 1, 2])
2  col = np.array([1, 2, 3])
```

인덱스 배열

```
1  X[row]
```

X에 인덱스(row) 전달

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

NumPy의 주요 기능

```
1 X[:, col]
```

```
array([[ 1,  2,  3],
       [ 5,  6,  7],
       [ 9, 10, 11]])
```

```
1 X[row, col]
```

```
array([ 1,  6, 11])
```

```
1 X[row.reshape(-1, 1), col]
```

Reshape로 인덱스 배열 구조를 변환해도 결과 동일

```
array([[ 1,  2,  3],
       [ 5,  6,  7],
       [ 9, 10, 11]])
```

NumPy의 주요 기능

NumPy 배열 객체와 복합 인덱싱

```
1 X = np.zeros(12).reshape((3, 4))
2 X
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
1 X[1, 0] = 1
2 X
```

```
array([[0., 0., 0., 0.],
       [1., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
1 X[1, [1, 3]] = 1
2 X
```

```
array([[0., 0., 0., 0.],
       [1., 1., 0., 1.],
       [0., 0., 0., 0.]])
```

NumPy의 주요 기능

```
1 X[[0, 2], [1, 3]] = 2
2 X
```

```
array([[0., 2., 0., 0.],
       [1., 1., 0., 1.],
       [0., 0., 0., 2.]])
```

```
1 X[0:3, [0, 2]] = 3
2 X
```

```
array([[3., 2., 3., 0.],
       [3., 1., 3., 1.],
       [3., 0., 3., 2.]])
```

NumPy의 주요 기능

NumPy 배열 객체의 정렬

```
1  np.random.seed(0)
2
3  x = np.array(np.random.randint(10, size=5))
4  x
```

```
array([5, 0, 3, 3, 7])
```

```
1  np.sort(x)
```

np.sort(x) : x를 순서대로 정렬한 배열을 반환 (x 원본은 바뀌지 않음)

```
array([0, 3, 3, 5, 7])
```

```
1  x
```

```
array([5, 0, 3, 3, 7])
```

```
1  x.sort()
```

x.sort() : x를 순서대로 정렬(x 원본이 변경)

```
1  x
```

```
array([0, 3, 3, 5, 7])
```

NumPy의 주요 기능

```
1  np.random.seed(0)
2
3  x = np.array(np.random.randint(10, size=5))
4  x
```

```
array([5, 0, 3, 3, 7])
```

```
1  idx = np.argsort(x)
2  idx
```

np.argsort(x) : x의 sort된 인덱스를 반환

```
array([1, 2, 3, 0, 4], dtype=int64)
```

```
1  x[idx]
```

팬시 인덱싱으로 정렬 가능

```
array([0, 3, 3, 5, 7])
```

Pandas 소개

- NumPy를 기반으로 개발된 패키지
- 유연한 인덱스를 가진 1차원 배열의 구조의 Series 객체와 유연한 행 인덱스와 열 인덱스를 가진 2차원 배열의 구조의 DataFrame 객체를 제공
- 강력한 데이터 연산 기능을 제공
- DataFrame 객체는 여러 데이터 타입을 사용할 수 있으며, 값의 누락 역시 허용

Pandas의 주요 기능

Pandas 패키지와 Series 객체

```
1  import pandas as pd
```

pandas를 import할 때 일반적으로 사용하는 별칭

```
1  data = pd.Series(np.linspace(0, 1, num=5))
2  data
```

pd.Series로 데이터 구성

index

0	0.00
1	0.25
2	0.50
3	0.75
4	1.00

value

```
dtype: float64
```

```
1  data.values
```

series 객체의 values는 numpy.ndarray 타입

```
array([0.  , 0.25, 0.5 , 0.75, 1.  ])
```

```
1  type(data.values)
```

```
numpy.ndarray
```

```
1  data.index
```

```
RangeIndex(start=0, stop=5, step=1)
```

Pandas의 주요 기능

Series에서도 numpy 기반의 인덱싱, 슬라이싱, 마스킹, 팬시 인덱싱 사용 가능

```
1  data[1]
```

```
0.25
```

```
1  data[2:4]
```

```
2    0.50
3    0.75
dtype: float64
```

```
1  data[(data > 0.1) & (data < 0.6)]
```

```
1    0.25
2    0.50
dtype: float64
```

```
1  data[[2, 4]]
```

```
2    0.5
4    1.0
dtype: float64
```

Pandas의 주요 기능

```
1  list(data.keys())
```

Series.keys() : index를 반환

```
[0, 1, 2, 3, 4]
```

```
1  list(data.items())
```

Series.itmes() : index와 value 쌍을 반환

```
[(0, 0.0), (1, 0.25), (2, 0.5), (3, 0.75), (4, 1.0)]
```

```
1  data.index = ["a", "b", "c", "d", "e"]
2  data
```

Index를 숫자가 아닌 값으로 설정 가능

적
inde

```
a    0.00
b    0.25
c    0.50
d    0.75
e    1.00
dtype: float64
```

```
1  data.index
```

```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

Pandas의 주요 기능

Series 객체와 loc 인덱서, iloc 인덱서의 활용

```
1  data.loc["a"]
```

```
0.0
```

loc 인덱서 : 명시적 인덱스에 대해 사용
iloc : 암묵적 인덱스에 대해 사용

```
1  data.loc["a":"c"]
```

"a"에서 "c"까지

```
a    0.00
b    0.25
c    0.50
dtype: float64
```

```
1  data.loc[["a", "c"]]
```

```
a    0.0
c    0.5
dtype: float64
```

```
1  data.loc[data > 0.7]
```

```
d    0.75
e    1.00
dtype: float64
```

Pandas의 주요 기능

```
1  data.iloc[0]
```

```
0.0
```

```
1  data.iloc[0:3]
```

```
a    0.00
b    0.25
c    0.50
dtype: float64
```

0~2까지 → "a"에서 "c"까지
(슬라이싱의 경우 암묵적 인덱스 활용 시 end 인덱스는 미포함됨.
loc와 iloc의 차이에 주의)

```
1  data.iloc[[0, 2]]
```

```
a    0.0
c    0.5
dtype: float64
```

Pandas의 주요 기능

Pandas 패키지와 DataFrame 객체

DataFrame : 행 인덱스와 열 이름으로 구성
(다양한 타입의 데이터를 사용 가능)

```
1  np.random.seed(0)
2
3  df = pd.DataFrame(np.random.randint(10, size=(3, 4)), columns=['col1', 'col2', 'col3', 'col4'])
4  df
```

DataFrame 객체의 values는 numpy.ndarray 타입

	col1	col2	col3	col4
0	5	0	3	3
1	7	9	3	5
2	2	4	7	6

```
1  df["col2"]
```

column의 이름을 key로 접근하는 방법

```
0    0
1    9
2    4
Name: col2, dtype: int32
```

```
1  df.col2
```

column의 이름을 key로 접근하는 방법

```
0    0
1    9
2    4
Name: col2, dtype: int32
```

Pandas의 주요 기능

DataFrame객체와loc인덱서, iloc인덱서의활용

```
df.loc[0, "col2":"col3"]
```

Series 객체를 반환

```
col2    0
col3    3
Name: 0, dtype: int32
```

```
df.loc[0:0, "col2":"col3"]
```

DataFrame 객체를 반환

	col2	col3
0	0	3

```
df.loc[[0], "col2":"col3"]
```

DataFrame 객체를 반환

	col2	col3
0	0	3

Pandas의 주요 기능

```
1  df.loc[0:2, "col2":"col3"]
```

loc에서는 0,1,2행 반환

	col2	col3
0	0	3
1	9	3
2	4	7

```
1  df.loc[(df["col2"] > 2) & (df["col3"] < 5), "col2":"col3"]
```

	col2	col3
1	9	3

Pandas의 주요 기능

```
1  df.iloc[0, 1:3]
```

```
col2    0
col3    3
Name: 0, dtype: int32
```

iloc에서는 1,2행 반환
(end 인덱스 미포함)

```
1  df.iloc[0:1, 1:3]
```

	col2	col3
0	0	3

```
1  df.iloc[[0], 1:3]
```

	col2	col3
0	0	3

Pandas의 주요 기능

```
1  df.iloc[0:3, 1:3]
```

	col2	col3
0	0	3
1	9	3
2	4	7

Pandas의 주요 기능

DataFrame객체의열 추가

```
1  df
```

	col1	col2	col3	col4
0	5	0	3	3
1	7	9	3	5
2	2	4	7	6

새로운 열 추가

```
1  df["total"] = df.sum(axis=1)
2  df
```

	col1	col2	col3	col4	total
0	5	0	3	3	11
1	7	9	3	5	24
2	2	4	7	6	19

Pandas의 주요 기능

DataFrame 객체의 행과 열 제거

```
1  df = df.drop(columns=["col4", "total"], axis=1)
2  df
```

열 제거

	col1	col2	col3
0	5	0	3
1	7	9	3
2	2	4	7

```
1  df.drop(index=1, axis=0)
```

행 제거

	col1	col2	col3
0	5	0	3
2	2	4	7

Pandas의 주요 기능

DataFrame객체의 널 값 연산

```
1  df = pd.DataFrame([[1, 2, 3],
2                     [4, 5, 6],
3                     [np.nan, 8, 9],
4                     [10, np.nan, 12]])
5  df
```

np.nan : 널 값(존재하지 않는 값)
(nan : not a number)

	0	1	2
0	1.0	2.0	3
1	4.0	5.0	6
2	NaN	8.0	9
3	10.0	NaN	12

```
1  df.dropna(axis=0)
```

df.dropna() : NaN값이 존재하는 행 또는 열 제거

	0	1	2
0	1.0	2.0	3
1	4.0	5.0	6

Pandas의 주요 기능

```
1  df.fillna(df.mean(axis=0))
```

df.fillna(value) : NaN값을 특정 값으로 대체

	0	1	2
0	1.0	2.0	3
1	4.0	5.0	6
2	5.0	8.0	9
3	10.0	5.0	12

Pandas의 주요 기능

DataFrame객체의조인

```
df1 = pd.DataFrame({'name': ['이순신', '강감찬', '을지문덕', '김유신'],
                    'dept': ['연구개발', '영업', '연구개발', '인사']})
df2 = pd.DataFrame({'emp_name': ['강감찬', '을지문덕', '이순신', '이순신'],
                    'project': ["S", "D", "A", "S"]})
```

```
pd.merge(df1, df2, left_on="name", right_on="emp_name").drop("emp_name", axis=1)
```

중복되는 열을 제거하기 위함

	name	dept	project
0	이순신	연구개발	A
1	이순신	연구개발	S
2	강감찬	영업	S
3	을지문덕	연구개발	D

pd.merge() : 두 DataFrame을 동일한 값을 가진 colum값(left_on, right_on)을 기준으로 연결

Pandas의 주요 기능

```
1  pd.merge(df1, df2, how="outer", left_on="name", right_on="emp_name").drop("emp_name", axis=1)
```

outer join 키워드
(Join이 안된 행을 누락시키지 않도록 할 때 사용)

	name	dept	project
0	이순신	연구개발	A
1	이순신	연구개발	S
2	강감찬	영업	S
3	을지문덕	연구개발	D
4	김유신	인사	NaN

Pandas의 주요 기능

DataFrame객체의정렬

```
1 import seaborn as sns
2 titanic = sns.load_dataset("titanic")
3 titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	em
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	S
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	S
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	S
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	S

Pandas의 주요 기능

DataFrame객체의정렬

운임(내림차순), 성별(오름차순)으로 정렬

head() : 데이터프레임의 앞부분 일부만 발췌
tail() : 데이터프레임의 끝부분 일부만 발췌

```
1 titanic.sort_values(by=["fare", "sex"], ascending=[False, True]).head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck
258	1	1	female	35.0	0	0	512.3292	C	First	woman	False	NaN
679	1	1	male	36.0	0	1	512.3292	C	First	man	True	B
737	1	1	male	35.0	0	0	512.3292	C	First	man	True	B
88	1	1	female	23.0	3	2	263.0000	S	First	woman	False	C
341	1	1	female	24.0	3	2	263.0000	S	First	woman	False	C

Pandas의 주요 기능

DataFrame객체의 그룹 연산

그룹화할 컬럼 지정 | 그룹 연산할 컬럼 | 그룹 연산 방법

```
1 titanic.groupby("sex")[["survived"]].aggregate("mean")
```

	survived
sex	
female	0.742038
male	0.188908

apply() 함수를 이용해 결과값을 가공처리

```
1 titanic.groupby("sex")[["survived"]].aggregate("mean").apply(lambda x: x - x.mean())
```

평균을 0으로 표준화

	survived
sex	
female	0.276565
male	-0.276565

Pandas의 주요 기능

DataFrame객체와피벗테이블

```
1 titanic.groupby(["sex", "class"])["survived"].aggregate("mean").unstack()
```

[ ] → Series로 반환

Series를 DataFrame 으로 반환

Series의 index가 컬럼으로 변환

sex	class	First	Second	Third
female		0.968085	0.921053	0.500000
male		0.368852	0.157407	0.135447

```
1 titanic.pivot_table("survived", index="sex", columns="class")
```

피벗테이블 생성

sex	class	First	Second	Third
female		0.968085	0.921053	0.500000
male		0.368852	0.157407	0.135447