Week 1_예습과제_김정은

1. Numpy

- C/C++ 과 같은 저수준 언어 기반의 호환 API 제공
- 배열 기반의 연산은 물론, 다양한 데이터 핸들링 기능을 제공
- 많은 머신러닝 기능들이 넘파이를 기반으로 작성되어 있음, 따라서 중요

(1) ndarray

- 넘파이 모듈 임포트

```
import numpy as np
```

- shape, ndarray 배열의 차원 알아보기 사용 예제

```
#ndarray를 만들어보고, 그 타입
array1 = np.array([1,2,3])
print('array1 type: ', type(array1))
print('array1 array 형태: ', array1.shape)
array2 = np.array([[1,2,3], [2,3,4]])
print('array2 type: ', type(array2))
print('array2 array 형태: ', array2.shape)
array3 = np.array([[1,2,3]])
print('array3 type: ', type(array3))
print('array3 array 형태: ', array3.shape)
```

output

```
array1 type:  <class 'numpy.ndarray'>
array1 array 형태:  (3,)
array2 type:  <class 'numpy.ndarray'>
array2 array 형태:  (2, 3)
```

```
array3 type:  <class 'numpy.ndarray'>
array3 array 형태:  (1, 3)
```

- ndarray.ndim(차원 확인)

```
print('array1: {0}차원, array2: {1}차원, array3: {2}차원'.form
at(array1.ndim, array2.ndim, array3.ndim))
```

output

```
array1: 1차원, array2: 2차원, array3: 2차원
```

- ndarray의 데이터 타입

```
list1 = [1,2,3]
print(type(list1))
array1 = np.array(list1)
print(type(array1))
print(array1, array1.dtype)
```

output

```
<class 'list'>
<class 'numpy.ndarray'>
[1 2 3] int32
```

- int형과 float형이 섞여있는 리스트를 ndarray로 변경시, 데이터 타입은 더 큰 타입인 float로 변경된다.

ndarray 생성 방법

1. arange() : array를 range()로 표현, 0부터 함수 인자값 -1까지를 순차적으로 데이터 값으로 변환

2. zero() : shape 형태 값을 입력하면 모든 값을 0으로 가진 해당 shape의 ndarray 반환
3. ones() : 위와 동일, 단 모든 값이 1.

Reshape
차원과 크기를 변경. 단, 지정된 사이즈로 변경이 불가능하면 오류가 발생.

```
array1 = np.arange(10)
print('array1:\\n', array1)
array2 = array1.reshape(2,5)
print('array2:\\n', array2)
array3 = array1.reshape(5,2)
print('array3:\\n', array3)
```

output

```
array1:
 [0 1 2 3 4 5 6 7 8 9]
array2:
 [[0 1 2 3 4]
 [5 6 7 8 9]]
array3:
 [[0 1]
 [2 3]
 [4 5]
 [6 7]
 [8 9]]
```

(2) Indexing

1. **특정 데이터 추출** : 인덱스 값 지정시 반환
2. **슬라이싱** : 연속된 인덱스 상의 ndarray 추출, ':' 기호 사이에 시작 인덱스, 종료 인덱스 표시하면 종료 인덱스 -1 위치에 있는 데이터의 ndarray 반환

3. **팬시 인덱싱** : 일정한 인덱싱 집합을 리스트나 ndarray로 지정, 해당 위치의 ndarray 반환
4. **불린 인덱싱** : True/False 값 인덱싱을 기반으로 True 위치의 인덱스 데이터의 ndarray 반환

1. 단일 값 추출 예시

```
#1부터 9까지의 1차원 ndarray 생성
array1 = np.arange(start=1, stop=10)
print('array1:', array1)
#index는 0부터 시작하므로 array1[2]는 3번째 index 위치의 데이터값을 의미
value = array1[2]
print('value:', value)
print(type(value))
```

output

```
array1: [1 2 3 4 5 6 7 8 9]
value: 3
<class 'numpy.int32'>
```

2. 슬라이싱 예시

```
array1 = np.arange(start=1, stop=10)
array4 = array1[:3]
print(array4)
array5 = array1[3:]
print(array5)
array6 = array1[:]
print(array6)
```

output

```
[1 2 3]
[4 5 6 7 8 9]
[1 2 3 4 5 6 7 8 9]
```

3. 팬시 인덱싱 예시

```
array1d = np.arange(start=1, stop=10)
array2d = array1d.reshape(3,3)
array3 = array2d[[0,1],2]
print('array2d[[0,1], 2] => ', array3.tolist())
array4 = array2d[[0,1], 0:2]
print('array2d[[0,1], 0:2] => ', array4.tolist())
array5 = array2d[[0,1]]
print('array2d[[0,1]] => ', array5.tolist())
```

output

```
array2d[[0,1], 2] =>  [3, 6]
array2d[[0,1], 0:2] =>  [[1, 2], [4, 5]]
array2d[[0,1]] =>  [[1, 2, 3], [4, 5, 6]]
```

4. 불린 인덱싱 예시

```
array1d = np.arange(start=1, stop=10)
#[ ] 안에 array1d > 5 Boolean indexing을 적용
array3 = array1d[array1d > 5]
print('array1d > 5 불린 인덱싱 결과값: ', array3)
```

output

```
array1d > 5 불린 인덱싱 결과값:  [6 7 8 9]
```

(3) 행렬의 정렬 : sort() 와 argsort()

sort

- 넘파이의 행렬 정렬은 np.sort()와 같이 넘파이 자체에서 sort()를 호출하는 방식, 그리고 ndarray.sort()와 같이 행렬 자체에서 sort()를 호출하는 방식으로 나뉜다.
- np.sort()의 경우 원행렬 유지, 정렬된 행렬 반환. 그리고 ndarray.sort()는 원행렬 정렬, 반환값은 None.

예시

```
#np.sort() 와 ndarray.sort() 의 차이점
org_array = np.array([3,1,9,5])
print('원본 행렬:', org_array)
#np.sort() 로 정렬
sort_array1 = np.sort(org_array)
print('np.sort() 호출 후 반환된 정렬 행렬:', sort_array1)
print('np.sort() 호출 후 원본 행렬:', org_array)
#ndarray.sort() 로 정렬
sort_array2 = org_array.sort()
print('org_array.sort() 호출 후 반환된 행렬:', sort_array2)
print('org_array.sort() 호출 후 원본 행렬:', org_array)
```

output

```
원본 행렬: [3 1 9 5]
np.sort() 호출 후 반환된 정렬 행렬: [1 3 5 9]
np.sort() 호출 후 원본 행렬: [3 1 9 5]
org_array.sort() 호출 후 반환된 행렬: None
org_array.sort() 호출 후 원본 행렬: [1 3 5 9]
```

- 기본적으로 sort()는 오름차순으로 행렬을 정렬한다. 내림차순으로 바꾸고 싶다면 [::-1]을 뒤에 삽입해준다.

argsort

- 원본 행렬이 정렬되었을 때, 기존 행렬의 원소에 대한 인덱스 값이 필요할 경우 사용

예시

```
#정렬된 행렬의 인덱스 방환(argsort)
org_array = np.array([3,1,9,5])
sort_indices = np.argsort(org_array)
print(type(sort_indices))
print('행렬 정렬 시 원본 행렬의 인덱스:', sort_indices)
```

output

```
<class 'numpy.ndarray'>
행렬 정렬 시 원본 행렬의 인덱스: [1 0 3 2]
```

(4) 선형대수 연산

1. 행렬 내적, 곱

- dot() 함수를 이용한다.

예시

```
#행렬 내적
A = np.array([[1,2,3], [4,5,6]])
B = np.array([[7,8], [9,10], [11,12]])
dot_product = np.dot(A,B)
print('행렬 내적 결과:\\n', dot_product)
```

output

```
행렬 내적 결과:
 [[ 58  64]
 [139 154]]
```

2. 전치행렬

- transpose() 함수를 이용한다.

예시

```
#전치 행렬
A = np.array([[1,2], [3,4]])
transpose_mat = np.transpose(A)
print('A의 전치 행렬:\\n', transpose_mat)
```

output

```
A의 전치 행렬:
 [[1 3]
 [2 4]]
```

2. Pandas

- 파이썬에서 데이터 처리를 위해 존재하는 라이브러리
- 2차원, 행과 열로 이루어져있는 데이터를 가공/처리할 수 있는 기능들을 제공
- DataFrame을 핵심 개체로 사용

Index : RDBMS의 PK 처럼 개별 데이터를 고유하게 식별하는 KEY 값

Series : index를 가지며 칼럼이 하나뿐인 데이터구조체

DataFrame : index를 가지며 칼럼이 여러 개인 데이터 구조체

- pandas 불러오기

```
import pandas as pd
```

- Kaggle에서 타이타닉 데이터를 불러오고, 데이터 출력 (head → 앞의 3줄만 출력)

```
titanic_df = pd.read_csv('train.csv')
print('titanic 변수 type:', type(titanic_df))
titanic_df.head(3)
```

- **shape, info(), describe()** : 데이터의 정보를 확인하는 데 유용.
  shape → 몇 행, 몇 열
  info() → 총 데이터 건수, 데이터 타입, Null 건수
  describe() → 칼럼별로 숫자형 데이터값들의 n-percentile 분포도, 평균값, 최댓값, 최솟값

- **Value_counts** : 지정된 칼럼의 데이터값 건수를 반환. 데이터 분포를 확인하는 데 유용

```
value_counts = titanic_df['Pclass'].value_counts()
print(type(value_counts))
print(value_counts)
```

output

```
<class 'pandas.core.series.Series'>
3    491
1    216
2    184
Name: Pclass, dtype: int64
```

기본적으로, value_counts 는 Null 값을 고려하지 않는다.
Null 값을 고려하고 싶다면,
value_counts(dropna=False) 로 바꿔준다.

** 넘파이, 리스트, 딕셔너리 <-> DataFrame 은 상호 전환이 가능하다.**

- **칼럼 데이터 세트 생성과 수정** : 연산자 사용

생성 예시

```
#칼럼 데이터 셋 생성과 수정
titanic_df['Age_0']=0
titanic_df.head(3)
```

수정 예시

```
titanic_df['Age_by_10'] = titanic_df['Age']*10
titanic_df['Family_No'] = titanic_df['SibSp']+titanic_df['P
arch']+1
titanic_df.head(3)
```

- DataFrame 데이터 삭제 : drop() 메서드 사용

예시

```
#칼럼 삭제
titanic_drop_df = titanic_df.drop('Age_0', axis=1)
titanic_drop_df.head(3)
```

이때, axis=0은 로우, axis=1은 칼럼을 뜻한다

inplace : **False** 이면 원본 DataFrame은 삭제 x, 삭제된 결과 DataFrame 반환. **True** 이면 원본 DataFrame에서 삭제.

Index 개체

- 객체 추출, array로 변환 : array변환시, .values 사용

```
#index 추출
indexes = titanic_df.index
#index 객체를 실제값 array로 변환
print('Index 객체 array값:\\n', indexes.values)
```

index 개체에 슬라이싱 사용 가능, index는 연산에서 사용되지 않고 오직 식별용으로만 사용된다.

- DataFrame 및 Series에 reset_index() 메서드 → 새롭게 인덱스를 연속 숫자형으로 할당하고, 기존 인덱스는 'index'라는 새로운 칼럼명으로 추가

데이터 셀렉션 및 필터링

DataFrame의 [] 연산자 → 칼럼만 지정할 수 있는 '칼럼 지정 연산자'로 이해

- 칼럼명의 이해 코드

```
#데이터 셀렉션 및 필터링
#DataFrame [] 연산자 -> 칼럼만 지정할 수 있는'칼럼 지정 연산자'
#0은 칼럼명이 아니기 때문에 오류 발생
print('단일 칼럼 데이터 추출:\n', titanic_df['Pclass'].head(3))
print('\n여러 칼럼의 데이터 추출:\n', titanic_df[['Survived', 'Pc]
print('[ ] 안에 숫자 index는 KeyError 오류 발생:\n', titanic_df[0
```

- 판다스의 인덱스 형태로 변환 가능한 표현식은 []내에 입력 가능
- 불린 인덱싱 표현도 가능

- DataFrame iloc[] 연산자 예제

```
#DataFrame iloc[] 연산자
#iloc[]은 위치 기반 인덱싱만 허용, 행과 열의 좌표 위치에 해당하는 값으로
data = {'Name':['Chulmin', 'Eunkyung', 'Jinwoong', 'Soobeom']
       'Year': [2001, 2016, 2015, 2015],
       'Gender': ['Male', 'Female', 'Male', 'Male']
       }
data_df = pd.DataFrame(data, index=['one', 'two', 'three', 'f
data_df
```

- 첫번쨰 행, 첫번째 열의 데이터 -> iloc[]을 통해 추출 가능
- iloc[]에 데이터프레임의 인덱스 값이나 칼럼명을 입력하면 오류가 발생한다!
- iloc[] 의 행 위치에 데이터프레임의 인덱스인 'one'이 들어가면 오류가 발생한다!
- iloc[]은 열 위치에 -1을 입력 -> 데이터프레임의 가장 마지막 열 데이터를 가져온다
- iloc[] 슬라이싱과 팬시 인덱싱은 제공, 그러나 불린 인덱싱은 제공X

- DataFrame loc[] 연산자(명칭 기반으로 데이터 추출한다) 예제

```
#DataFrame loc[] 연산자(명칭 기반으로 데이터 추출)
data_df.loc['one', 'Name']
```

Output ‘Chulmin’

- 주의!! 첫번째 행 위치의 인덱스 값을 0으로 착각하고 행 위치에 0을 입력 -> 오류 발생
- data_df는 0을 인덱스로 가지고 있지 않는다.
- loc[]에 슬라이싱 기호를 적용하면 종료값까지 포함(명칭 기반 인덱싱의 특성)

- **불린 인덱싱 예시**

```
# 타이타닉 데이터셋 다시 로딩, 나이가 60세 이상인 데이터만을 추출
titanic_df = pd.read_csv('train.csv')
titanic_boolean = titanic_df[titanic_df['Age'] > 60]
print(type(titanic_boolean))
titanic_boolean
```

- **여러 조건들로 승객들 구분 추출 예시**

```
# 60세 이상인 승객의 나이와 이름만 추출
titanic_df[titanic_df['Age']>60][['Name', 'Age']].head(3)

# loc을 이용해도 동일하게 적용 가능(단, ['Name', 'Age']는 칼럼 위치에)
titanic_df.loc[titanic_df['Age']>60, ['Name', 'Age']].head(3)

# 나이가 60세 이상이고 선실 등급이 1등급, 성별이 여성인 승객 추출
titanic_df[ (titanic_df['Age']>60) & (titanic_df['Pclass']==1

# 개별 조건 변수 할당, 변수를 결합해 불린 인덱싱 수행
cond1 = titanic_df['Age']>60
cond2 = titanic_df['Pclass']==1
```

```
cond3 = titanic_df['Sex']=='female'
titanic_df[ cond1 & cond2 & cond3 ]
```

정렬, Aggregation 함수, GroupBy 적용

- DataFrame, Series의 정렬은 sort_values()를 사용하면 편리하다.

```
#예시
#titanic_df를 Name 칼럼으로 오름차순 정렬
titanic_sorted = titanic_df.sort_values(by=['Name'])
titanic_sorted.head(3)

#여러 개의 칼럼으로 정렬 -> by에 리스트 형식으로, 정렬하려는 칼럼 입력
titanic_sorted = titanic_df.sort_values(by=['Pclass', 'Name']
titanic_sorted.head(3)

#특정 칼럼에 aggregation 함수를 적용하기 위해, 대상 칼럼들만 추출
titanic_df[['Age', 'Fare']].mean()

#groupby() 적용
#Pclass 칼럼 기준으로 GroupBy된 DataFrameGroupby 객체 반환
titanic_groupby = titanic_df.groupby(by='Pclass')
print(type(titanic_groupby))

#DataFrame에 groupby() 호출, 반환된 결과에 aggregation 함수 호출
#groupby() 대상 칼럼을 제외한 모든 칼럼에 해당 aggregation 함수를 적용
titanic_groupby = titanic_df.groupby('Pclass').count()
titanic_groupby

#PassengerId와 Survived 칼럼에만 count() 시행
titanic_groupby = titanic_df.groupby('Pclass')[['PassengerId'
titanic_groupby

# 서로 다른 aggregation 함수를 적용할 경우, agg() 내에 인자로 입력
titanic_df.groupby('Pclass')['Age'].agg([max, min])

# agg() 내에 입력값으로 딕셔너리 형태로 aggregation이 적용될 칼럼들과 a
```

```
agg_format={'Age':'max', 'SibSp':'sum', 'Fare':'mean'}
titanic_df.groupby('Pclass').agg(agg_format)
```

결손 데이터 처리

isna() 또는 fillna()를 이용한다.

- 결손 데이터의 개수는 isna() 결과에 sum() 함수를 추가해 구할 수 있다.
- fillna()로 결손 데이터를 대체할 수 있다.
- 그런데 fillna를 사용할 경우, inplace=True 파라미터를 추가해야 데이터 셋 값 변형된다.

apply lambda 식으로 데이터 가공

- lambda는 함수의 선언과 함수 내의 처리를 한줄의 식으로 쉽게 변환하는 도구이다.
- lambda 식을 이용할 때, 여러 개 값을 인자로 사용해야 할 경우, map() 함수를 결합하면 좋다.
- 다만, if, else만 지원하고 if, else if, else와 같이 else if는 지원하지 않는다. → 다른 처리가 필요하다.(해당 코드 아래쪽)

```
titanic_df['Age_cat'] = titanic_df['Age'].apply(lambda x: 'Ch
titanic_df['Age_cat'].value_counts()
```