

1주차 예습 과제

머신러닝의 개념

- 머신러닝: 애플리케이션을 수정하지 않고도 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법
 - 지도학습, 비지도학습, 강화학습으로 나뉨
 - 데이터에 매우 의존적 → 가비지 인, 가비지 아웃

파이선 머신러닝 생태계를 구성하는 주요 패키지

- 머신러닝 패키지: 사이킷런
- 행렬/선대/통계 패키지: 넘파이
- 데이터 핸들링: 판다스 → 2차원 데이터 처리에 특화
- 시각화: 맷플롯립, 시본

넘파이

- Numpy=Numerical Python
 - 루프 없이 대량 데이터의 배열 연산을 가능하게 하여 빠르다
 - C/C++ 등 저수준 언어 기반의 호환 API를 제공한다
 - 데이터 핸들링 기능도 제공하지만 판다스의 편리성에는 미치지 못한다
- 기반 데이터 타입은 ndarray → 다차원 배열을 쉽게 생성하고 연산 수행 가능
 - 숫자 값(int, unsigned int, float, complex), 문자열 값, 불 값 모두 가능
 - 하지만 한개의 객체에는 **한 가지 데이터 타입만 가능**
- `import numpy as np` : 넘파이 모듈 임포트
- `np.ndarray()` : 리스트 등 인자를 입력받아 ndarray로 변환
- `ndarray.shape` : ndarray의 행과 열의 수를 튜플 형태로 가짐
 - (2,3): 2차원, 로우 2개, 칼럼 3개, 6개의 데이터
 - (3,) : 1차원, 3개의 데이터 **vs** (1,3): 2차원, 로우 1개, 칼럼 3개

- `ndarray.ndim` : 차원 확인
- `ndarray.dtype` : ndarray내의 데이터 타입 확인
 - 다른 유형이 섞여있는 리스트를 ndarray로 변경하면 데이터 크기가 더 큰 dtype으로 일괄 적용
- `ndarray.astype('...')` : ndarray 내 데이터값의 타입 변경
- 특정 크기의 ndarray를 연속값/0/1로 쉽게 생성할 때:
 - `arange(start,stop)` : start부터 (stop -1)까지 순차적으로 데이터값이 됨
 - default start 값 = 0
 - `zeros(..., dtype='...')` : 모든 값이 0인 해당 shape의 ndarray 생성
 - default dtype=float64
 - `ones(..., dtype='...')` : 모든 값이 1인 해당 shape의 ndarray 생성
 - default dtype=float64
- `ndarray.reshape(...)` : 특정 차원 및 크기로 변환
 - -1을 인자로 사용하면 호환되는 새로운 크기로 변환해줌
 - 지정된 사이즈로 변경이 불가능하면 오류 발생
- 인덱싱:
 - `ndarray[...]` : 특정 데이터 추출 (원하는 위치의 인덱스 값을 지정)
 - 인덱스는 0부터 시작
 - 더 이상 ndarray 타입 아님
 - (-): 맨 뒤에서부터 데이터 추출 (ex. -1: 맨 뒤의 데이터 값)
 - 2차원 인덱스: **[row,col]**
 - axis0=로우 방향 축, axis1=컬럼 방향 축
 - `ndarray[...:...]` : **슬라이싱**, 시작 인덱스에서 종료 인덱스-1의 위치에 있는 데이터의 ndarray 를 반환
 - 시작 인덱스 생략 = 0으로 간주
 - 종료 인덱스 생략 = -1로 간주
 - 시작/종료 인덱스 생략 = 0:-1으로 간주

- 2차원 슬라이싱: `[::-1, ::-1]`
 - 한 축에만 슬라이싱 적용하고 다른 한축에는 단일 값 인덱스 적용 가능
 - 다차원 ndarray에서 특정 축의 인덱스를 지정하지 않으면 차원이 축소된다
- **팬시 인덱싱**: 일정한 인덱싱 집합을 지정해 해당 위치 데이터의 ndarray 반환
- `ndarray[조건]` : **불린 인덱싱**, T/F값 인덱싱 집합을 기반으로 T에 해당하는 위치의 데이터 반환
- 행렬 정렬:
 - `np.sort(ndarray)` : 원 행렬은 유지, 정렬된 행렬을 반환
 - `ndarray.sort()` : 원 행렬을 정렬한 형태로 변환, 반환값은 None
 - 기본적으로 오름차순으로 정렬, 내림차순으로 정렬하려면 `[::-1]` 붙이기
 - axis 축 값 설정하여 특정 방향으로 정렬 할 수 있음
 - `np.argsort()` : 정렬 행렬의 원본 행렬 인덱스를 ndarray형으로 반환
- 선형대수 연산:
 - `np.dot(A,B)` : 행렬 내적(행렬 곱)
 - 왼쪽 행렬 열 개수와 오른쪽 행렬 행 개수가 같아야 내적 연산이 가능함
 - `np.transpose(A)` : 전치행렬(행과 열 위치 교환)

데이터 핸들링 - 판다스

- 2차원 데이터를 효율적으로 가공/처리할 수 있는 기능 제공
 - DataFrame: 여러 행과 열로 이뤄진 2차원 데이터를 담는 데이터 구조체
 - Index: 개별 데이터를 고유하게 식별하는 Key 값
 - Series: Index와 단 하나의 칼럼으로 구성된 데이터 세트 (DataFrame은 여러 개의 Series로 이뤄짐)
- 다양한 포맷으로 된 파일을 쉽게 DataFrame으로 로딩할 수 있다
 - `read_csv()`: csv 파일 포맷 변환을 위한 API
 - csv: 칼럼을 ','로 구분한 파일 포맷

- 어떤 필드 구분 문자 기반의 파일 포맷도 변환 가능
- `read_csv('파일명', sep='\t')`
- `read_fwf()`: 고정 길이 기반의 칼럼 포맷 변환을 위한 API

- `import pandas as pd` : 판다스 모듈 импорт
- `pd.read_csv()` : 파일명 인자로 들어온 파일을 로딩해 DataFrame 객체로 반환
 - 첫 번째 로우가 칼럼명으로 할당 됨
 - 고유의 Index 객체 값이 지정됨

```
#구글 코랩으로 파일 불러오는 법
from google.colab import drive
drive.mount('/content/gdrive')
```

```
directory_path = '/content/gdrive/My Drive/Colab Notebooks/Euron/'
df_train = pd.read_csv(directory_path + 'train.csv')
```

- `DataFrame.head()` : 맨 앞 N개의 로우를 반환 (default: 5개)
- `DataFrame.shape` : 행과 열을 튜플 형태로 반환
- `DataFrame.info()` : 총 데이터 수, 칼럼 별 데이터 타입, Null 수 알 수 있음
 - RangeIndex: index의 범위를 나타냄→전체 row 수 알 수 있음

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age           714 non-null float64
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
```

1 RangeIndex는 DataFrame index의 범위를 나타내므로 전체 row 수를 알 수 있습니다. 전체 데이터는 891개 row입니다. 그리고 Column 수는 12개입니다.

2 칼럼별 데이터 타입을 나타냅니다. 가령 Pclass 칼럼의 데이터 타입은 int64형입니다. Name 칼럼의 데이터 타입은 object 인데, 이는 문자열 타입으로 생각해도 무방합니다.

3 몇 개의 데이터가 non-null(Null 값이 아님)인지 나타냅니다. 가령 Age 칼럼의 714 non-null의 의미는 Age 칼럼 891개 데이터 중 714개가 Null이 아니며 177개는 Null이라는 의미입니다.

4 전체 12개의 칼럼들의 타입을 요약한 것입니다. 2개 칼럼이 float64, 5개 칼럼이 int64, 5개 칼럼이 object 타입입니다.

- `DataFrame.describe()` : 칼럼 별 숫자형 데이터값(int, float 등)의 기술 통계 요약
 - object 타입의 칼럼은 제외시킴

- Not Null인 데이터 건수, 평균, 표준편차, 백분위수, 최대/최소값 나타냄
- 숫자 칼럼이 숫자형 카테고리(범주형) 칼럼인지 판단하게 도와줌
- `DataFrame['칼럼명']` : Series 형태로 특정 칼럼의 데이터들이 반환됨
- `Series.value_counts()` : 데이터값 건수를 series 객체로 반환, 건수가 많은 순으로 정렬됨
- `dropna=True` : Null값을 무시하여 건수를 계산 (default=True)
- Index 값:
 - DataFrame, Series가 만들어진 후에도 변경할 수 있다.
 - 숫자형뿐만 아니라 문자열도 가능함
 - 단 고유성이 보장되어야 한다.
- DataFrame과 리스트, 딕셔너리, 넘파이 ndarray는 상호 변환이 가능함
 - `pd.DataFrame(__, columns=__)` : DataFrame으로 변환
 - 딕셔너리를 DataFrame으로 변환 시 key는 칼럼명(문자열 형태), value는 해당 칼럼 데이터(리스트 형태)로 변환됨
 - `DataFrame.values` : DataFrame을 ndarray로 변환
 - `DataFrame.values.tolist()` : DataFrame을 list로 변환
 - `DataFrame.to_dict('list')` : DataFrame을 딕셔너리로 변환
 - 인자로 'list' 입력하면 딕셔너리 값이 리스트형으로 반환
- `DataFrame['칼럼명']=값` : 새 칼럼 데이터 생성 / 기존 칼럼 값 업데이트
 - 상수값 할당 시 모든 데이터 세트에 일괄 적용됨
 - 기존 칼럼 Series를 가공해 새로운 칼럼 Series를 추가할 수 있음
- `DataFrame.drop(labels='__', axis=__, inplace=__)` : 데이터 삭제
 - `axis=0` 이면 로우를 드롭 → labels에 오는 값을 인덱스로 간주
 - `axis=1` 이면 칼럼을 드롭 → labels에 오는 값을 칼럼명으로 간주
 - `inplace=True` : 원본 DataFrame의 데이터를 삭제 (default=False)
 - 이때 반환값은 None
- Index 객체:
 - `DataFrame.index` / `Series.index` : Index 객체만 추출
 - `index.values` : index 객체에 들어있는 실제 값 (1차원 ndarray)

- 단일 값 반환 및 슬라이싱도 가능
- Index 객체는 **수정 불가능**
- Series 객체에 연산 함수를 적용할 때 Index는 연산에서 제외된다.
- `DataFrame.reset_index(inplace=___)` : 인덱스를 연속 숫자형으로 할당
 - 기존 인덱스는 'index'라는 새로운 칼럼으로 추가됨
 - `drop=True` : 기존 인덱스가 추가되지 않고 삭제됨
- 데이터 셀렉션 및 필터링:
 - `DataFrame[]` : 칼럼명 문자, 인덱스로 변환 가능한 표현식, 불린 인덱싱 표현만 입력 가능
 - `DataFrame.iloc[,]` : 위치 기반 인덱싱
 - `DataFrame.loc[,]` : 명칭 기반 인덱싱
 - 슬라이싱 기호(:) 적용하면 종료 값-1이 아니라 **종료값까지 포함**한다.
 - 불린 인덱싱 가능
- 불린 인덱싱:
 - `[] / loc[]` 에서 사용 가능
 - `& : and`
 - `| : or`
 - `~ : not`
- `DataFrame.sort_values(by=['___'],ascending=___, inplace=___)` : 정렬
 - `ascending=True/False` : 오름차순으로 정렬/내림차순 정렬 (default=True)
 - `inplace=True/False`: 원본 정렬 / 원본 유지, 정렬된 결과를 반환 (default=False)
- Aggregation 함수: `min()`, `max()`, `sum()`, `count()`
 - DataFrame에서 바로 호출할 경우 **모든 칼럼에 적용**함
 - 특정 칼럼만 적용하려면 대상 칼럼을 추출해서 적용해야 함.
(ex. `DataFrame[___].mean()`)
- `DataFrame.groupby(by='___')` : by에 입력된 칼럼으로 groupby됨.
 - **DataFrameGroupBy**라는 형태의 DataFrame 반환됨

- `groupby()`에 aggregation 함수를 호출하면 **groupby 대상 칼럼을 제외한 모든 칼럼**에 해당 함수를 적용함
- `DataFrame.groupby('___')['___']` : `groupby()`에 **특정 칼럼만** aggregation 함수를 적용하려면 해당 칼럼을 **필터링한 후** 적용해야 함.
- `DataFrame.groupby('___')['___'].agg([___,___])` : 여러 개의 aggregation 함수를 적용하려면 `agg()`에 인자로 입력하면 됨.
- 결손 데이터: 값이 없는, 즉 NULL인 경우 → NaN으로 표시
- `DataFrame.isna()` : 모든 칼럼의 값이 NaN인지 True/False로 알려줌
 - `DataFrame.isna().sum()` : 결손 데이터의 개수 (True=1, False=0)
- `DataFrame.fillna('___')` : 결손 데이터를 다른 값으로 대체
 - 반환 값을 다시 받거나 `inplace=True`로 해야 원본 수정됨.
- `DataFrame[___].apply(lambda x : ___)` : 일괄적으로 복잡한 데이터 가공이 필요할 때 사용
 - `lambda x : ___` : 함수 선언과 함수 내의 처리를 한 줄로 쉽게 변환한 식
 - (:) 왼쪽: 입력 인자
 - (:) 오른쪽: 입력 인자의 계산식(반환 값)
 - `map(lambda x : __, [___])` : 여러 개의 값을 입력할 때 `map()` 함수를 결합함
 - `lambda x : 반환값 if 조건 else 반환값` : if절에서는 조건보다 반환 값을 먼저 기술해야 함
 - `else if`는 지원X → `else` 절에 다시 `if else`를 ()로 묶어서 넣음
 - 복잡한 경우 별도의 함수를 만들어 반환값으로 지정