『판다스(Pandas) 개요와 기본 API - 01』~『파이썬 기반의 머신러닝과 생태계 이해 Summary』요약

- **판다스(Pandas)**
  - 판다스는 파이썬에서 데이터 처리를 위해 존재하는 가장 인기 있는 라이브러리
  - 대부분의 데이터 시트는 **2차원 데이터**(인간이 보기에 가장 직관적인 데이터)
  - **행(Row) x 열(Column)로 구성**되어 있는 데이터를 효율적으로 가공/처리할 수 있는 다양하고 훌륭한 기능 제공
- **판다스의 주요 구성 요소 – Data Frame, Series, Index**
  - **DataFrame**(Column – Rows 2차원 데이터 set): NumPy와 다르게 Columns를 가지고 있음
  - **Series**(1개의 Column값으로만 구성된 1차원 데이터 set): Column명이 없음, Series가 여러 개면 DataFrame이 됨
  - **Index**(DataFrame, Series의 고유한 key값 객체): Column에 포함되지 않음(=차원을 이루지 않음), 각 row를 구별함
- **기본 API**
  - **read_csv()**: csv 파일을 편리하게 DataFrame으로 로딩, (+sep인자를 콤마(,)가 아닌 다른 분리자로 변경하여 다른 유형의 파일도 로드가 가능)
  - **head()**: DataFrame의 맨 앞 일부 데이터만 추출 ex) 파일명.head(5)는 5개의 행만 보여줌 (*모든 data는 ndarray로 표현됨)
  - **shape**: DataFrame의 행(Row)와 열(Column) 크기를 알려줌
  - **info()**: DataFrame내의 Column명, 데이터 타입, Null건수, 데이터 건수 정보를 제공
  - **describe()**: 데이터 값들의 평균, 표준편차, 4분위 분포도를 제공, 숫자형 컬럼들에 대해서 해당 정보를 제공
  - **Value_counts()**: 동일한 개별 데이터 값이 몇 건이 있는지 정보를 제공, 즉 개별 데이터 값의 분포도를 제공, (*주의할 점은 value_counts()는 Series객체에서만 호출될 수 있으므로 반드시 DataFrame을 단일 컬럼으로 입력하여 Series로 변환한 뒤 호출 해야함)
  - **Sort_values()**: by=정렬 Column, ascending=True 또는 False로 오름차순/내림차순으로 정렬 (+여러 row를 기준으로 정렬 가능)

- **DataFrame과 리스트, 딕셔너리, 넘파이 ndarray 상호 변환**
  - **리스트 -> DataFrame**: DataFrame생성 인자로 리스트 객체와 매핑되는 Column명들을 입력
    (df_list1 = pd. DataFrame(list, columns=col_name1))
  - **ndarray -> DataFrame**: DataFrame생성 인자로 ndarray와 매핑되는 Column명들을 입력
    (df_array1 = pd. DataFrame(array2, columns=col_name2))
  - **딕셔너리 -> DataFrame**: 딕셔너리의 키(key)로 Column명을 값(value)를 리스트 형식으로 입력
    (dict = {'col1':[1,11], 'col2':[2,22], 'col3'[3,33]}
    df_dict = pd.DataFrame(dict))
  - **DataFrame -> ndarray**: DataFrame 객체의 values 속성을 이용하여 ndarray로 변환
  - **DataFrame -> 리스트**: DataFrame 객체의 values 속성을 이용하여 먼저 ndarray로 변환 후 tolist()를 이용하여 리스트로 변환
  - **DataFrame -> 딕셔너리**: DataFrame 객체의 to_dict()를 이용하여 변환
- **DataFrame의 컬럼 데이터 셋 Access**

  DataFrame의 컬럼 데이터 세트 생성과 수정은 [ ] 연산자를 이용, 새로운 컬럼에 값을 할당하려면 DataFrame [ ] 내에 새로운 컬럼명을 입력하고 값을 할당
- **DataFrame 데이터 삭제(drop()사용)**

  < DataFrame.drop(**labels=None**, **axis=0**, index=None, columns=None, level=None, **inplace=None**, errors='raise') >
  - DataFrame의 row를 삭제할 때는 **axis=0**, column을 삭제할 때는 **axis=1**으로 설정
  - 원본 DataFrame은 유지, drop된 DataFrame을 새롭게 객체 변수로 받고 싶다면 **inplace=False(default)** 적용, 원본 DataFrame에 drop된 결과를 적용하고 싶다면 **inplace=True** 적용
- **Index**
  - 판다스의 index 객체는 RDBMS의 PK(Prime key)와 유사하게 **DataFrame, Series의 레코드를 고유하게 식별하는 객체**
  - DataFrame, Series에서 index객체만 추출하려면 **DataFrame.index** 또는 **Series.index** 속성을 사용
  - Series객체는 index객체를 포함하지만 Series객체에 연산 함수를 적용할 때 index는 연산에서 제외됨, index는 오직 식별용으로만 사용
  - DataFrame 및 Series에 **reset_index()** 메서드를 수행하면 index를 연속 숫자형으로 할당하여 기존 index는 'index'라는 새로운 column명으로 추가됨

- **데이터 Selection 및 Filtering – ix, loc[], iloc[]**
  - **명칭(label) 기반 인덱싱**은 column의 명칭을 기반으로 지정하는 방식, '**column 명**' 같이 명칭으로 열 위치를 지정하는 방식
  - **위치(position) 기반 인덱싱**은 0을 출발점으로 하는 가로축, 세로축 좌표 기반의 행과 열 위치를 기반으로 데이터를 지정, **따라서 행, 열 위치 값으로 정수 입력됨**
  - **ix[]**: 명칭 기반과 위치 기반 인덱싱을 함께 제공(디버깅할 때 기준 애매함 -> 곧 사라질 예정)
  - **loc[]**: 명칭 기반 인덱싱
  - **iloc[]**: 위치 기반 인덱싱
- **불린 인덱싱(Boolean indexing)**

  위치기반, 명칭기반 인덱싱 모두 사용할 필요없이 **조건식을 [] 안에 기입하여** 간편하게 필터링
- **Aggregation 함수**
  - **sum(), max(), min(), count()** 등의 함수는 DataFrame/Series에서 집합연산을 수행
  - DataFrame의 경우 DataFrame에서 바로 aggregation을 호출할 경우 모든 column에 해당 aggregation을 적용
- **DataFrame Group By**
  - DataFrame은 Group By 연산을 위해 **groupby()메소드**를 제공
  - Groupby() 메소드는 by 인자로 group by 하려는 column명을 입력 받으면 **DataFrameGroupBy객체**를 변환
  - 이렇게 변환된 DataFrameGroupBy객체에 **aggregation함수**를 수행
- **결손 데이터(Missing data) 처리하기**
  - **isna()**: DataFrame의 isna() 메소드는 주어진 column값들이 NaN인지 True/False값을 반환(NaN이면 True)
  - **fillna()**: missing data를 인자로 주어진 값으로 대체
- **판다스 apply lambda – 파이썬 lambda 식 이해**

  *lambda: 런타임에 생성해서 사용할 수 있는 간단한 익명함수, 함수가 생성된 곳에서만 필요한 1회용 함수
- **apply lambda 식으로 데이터 가공**
  - 판다스는 apply함수에 lambda식을 결합해 DataFrame이나 Series의 레코드별로 데이터를 가공하는 기능을 제공
  - 판다스의 경우 column에 일괄적으로 데이터 가공하는 것이 속도가 빠르나 복잡한 데이터 가공이 필

요한 경우에는 apply lambda를 이용함

- **판다스 summary**
  - 2차원 데이터 핸들링을 위해서는 판다스 사용이 효율적
  - 판다스는 매우 편리하고 다양한 데이터 처리 API를 제공하지만, 이를 다 알기에는 많은 시간과 노력이 필요함
  - 무작정 공부하기 보다는, 데이터 처리를 직접 수행해 보면서 문제에 부딪칠 때마다 판다스의 다양한 API를 찾아서 해결해 가는 것이 더 효과적 공부 방법