chap 1 - 파이썬 기반의 머신러닝과 생태계 이해

01 머신러닝의 개념

애플리케이션을 수정하지 않고도 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법

업무적으로 복잡한 조건/규칙들이 다양한 형태로 결합하고 시시각각 변하면서 도저히 소프트웨어 코드로 로직을 구성하여 이를 관통하는 일정한 패턴을 찾기 어려운 경우 머신러닝은 훌륭한 솔루션을 제공함

머신러닝은 이러한 문제를 데이터를 기반으로 숨겨진 패턴을 인지해 해결한다.

머신러닝 알고리즘은 데이터를 기반으로 통계적인 신뢰도를 강화하고 예측 오류를 최소화하기 위한 다양한 수학적 기법을 적용해 데이터 내의 패턴을 스스로 인지하고 신뢰도 있는 예측 결과를 도출해 낸다

데이터 분석 영역은 머신러닝 기반의 예측 분석(Predictive Analysis)으로 재편되고 있다

머신러닝의 분류

지도학습(Supervised Learning) , 비지도학습(Un-Supervised Learning), 강화학습(Reinforcement Learning)

지도학습 - 분류(Classification), 회귀(Regression), 추천 시스템, 시각/음성 감지/인지, 텍스트 분석, NLP

비지도학습 - 클러스터링, 차원 축소, 강화학습

데이터 전쟁

머신러닝의 가장 큰 단점은 데이터에 매우 의존적

Garbage In, Garbage Out - 즉 좋은 품질의 데이터를 갖추지 못한다면 머신러닝의 수행 결과도 좋을 수 없다

최적의 머신러닝 알고리즘과 모델 파라미터를 구축하는 능력도 중요하지만 데이터를 이해하고 효율적으로 가공, 처리, 추출해 최적의 데이터를 기반으로 알고리즘을 구동할 수 있도록 준비하는 능력이 더 중요할 수도 있다

파이썬과 R 기반의 머신러닝 비교

C/C++/JAVA 등 컴파일러 기반의 언어는 개발 생산성이 떨어지고 지원 패키지와 생태계가 활발하지 않다

R은 통계 전용 프로그램 언어

파이썬은 다양한 영역에서 사용되는 개발 전문 프로그램 언어

파이썬이 R에 비해 뛰어난 점은? - "파이썬은 소리 없이 프로그래밍 세계를 점령하고 있는 언어"

- 쉽고 뛰어난 개발 생산성
- 오픈 소스 계열의 전폭적 지원을 받고 있고 놀라울 정도로 많은 라이브러리로 인해 개발 시 높은 생산성 보장
- Interpreter Language의 특성상 속도는 느리지만, 뛰어난 확장성, 유연성, 호환성으로 인해 서버, 네트워크, 시스템, IoT, 데스크톱까지 다양한 영역에서 사용
- 머신러닝 애플리케이션과 결합한 다양한 애플리케이션 개발 가능
- 다양한 기업 환경으로의 확산 가능
- 유수의 딥러닝 프레임워크인 TensorFlow, Keras, PyTorch 등에서 파이썬 우선 정책으로 파이썬을 지원하고 있다

02 파이썬 머신러닝 생태계를 구성하는 주요 패키지

- 머신러닝 패키지: Scikit - Learn
- 행렬/선형대수/통계 패키지: 행렬과 선형대수를 다루는 패키지는 넘파이(NumPy)
- 데이터 핸들링: 판다스는 파이썬 세계의 대표적인 데이터 처리 패키지, 2차원 데이터 처리에 특화되어 있으며 넘파이보다 훨씬 편리하게 데이터 처리를 할 수 있는 많은 기능을 제공함, 맷플롯립(Matplotlib)을 호출해 쉽게 시각화 기능을 지원
- 시각화: 파이썬의 대표적인 시각화 패키지는 맷플롯립. 그러나 너무 세분화된 API로 익히기가 번거롭고 시각적인 디자인 부분에서도 투박한 면이 있다. 이를 보완 → 시본(Seaborn)
- 시본: 맷플롯립을 기반으로 만들었지만 판다스와의 쉬운 연동, 더 함축적인 API, 분석을 위한 다양한 유형의 그래프/차트 제공
- 아이파이썬(IPython, Interactive Python): 대화형 파이썬 툴 - 학교에서 학생들에게 설명하듯이 프로그래밍과 이에 대한 설명적인 요소를 결합했다는 뜻. 전체 프로그램에

서 특정 코드 영역별로 개별 수행을 지원하므로 영역별로 코드 이해가 매우 명확하게 설명될 수 있다

- 주피터 노트북(Jupyter Notebook): 대표적인 아이파이썬 지원 툴로 중요 코드 단위로 설명을 적고 코드를 수행해 그 결과를 볼 수 있게 만들어서 직관적으로 어떤 코드가 어떤 역할을 하는지 매우 쉽게 이해할 수 있도록 지원한다

03 넘파이

넘파이 (Numerical Python - NumPy): 파이썬에서 선형대수 기반의 프로그램을 쉽게 만들 수 있도록 지원하는 대표적인 패키지

루프를 사용하지 않고 대량 데이터이 배열 연산을 가능하게 하므로 빠른 배열 연산 속도를 보장한다 - 대량 데이터 기반의 과학과 공학 프로그램은 빠른 계산 능력이 매우 중요하기 때문에 넘파이에 의존하고 있다

C/C++과 같은 저수준 언어 기반의 호환 API를 제공한다 - 수행 성능이 중요한 부분은 C/C++ 기반의 코드로 작성하고 이를 넘파이에서 호출하는 방식으로 쉽게 통합할 수 있다

다양한 데이터 핸들링 기능을 제공한다 - 그러나 대표적인 데이터 처리 패키지인 판다스의 편리성에는 미치지 못한다

넘파이 ndarray 개요

넘파이 array() 함수는 파이썬의 리스트와 같은 다양한 인자를 입력받아서 ndarray로 변환하는 기능을 수행한다.

생성된 narray 배열의 shape 변수는 ndarray의 크기, 즉 행과 열의 수를 튜플 형태로 가지고 있으며 이를 통해 ndarray 배열의 차원까지 알 수 있다.

(튜플: 순서가 정해진 여러 개의 데이터를 묶어서 저장하는 자료 구조)

np.array()의 사용법 : ndarray로 변환을 원하는 객체를 인자로 입력하면 ndarray를 반환한다. ndarray.shape는 ndarray의 차원과 크기를 튜플 형태로 나타낸다

리스트 []은 1차원이고, 리스트의 리스트 [[ ]]는 2차원과 같은 형태

ndarray의 데이터 타입

ndarray내의 데이터값은 숫자 값, 문자열 값, 불 값 등이 모두 가능하다

ndarray내의 데이터 타입은 그 연산의 특성상 같은 데이터 타입만 가능하다. ex) 한 개의 ndarray 객체에 int와 float가 함께 있을 수 없다

ndarray 내의 데이터 타입은 dtype 속성으로 확인할 수 있다

만약 다른 데이터 유형이 섞여 있는 리스트를 ndarray로 변경하면 데이터 크기가 더 큰 데이터 타입으로 형 변환을 일괄 적용한다. ex) int형과 string형이 섞여있을 때는 모두 string으로 변환된다

ndarray 내 데이터값의 타입 변경도 astype() 매서드를 이용해 할 수 있다.

astype()에 인자로 원하는 타입을 문자열로 지정하면 된다. 보통 메모리를 더 절약해야할 때 이용된다

ndarray를 편리하게 생성하기 - arange, zeros, ones

테스트용으로 데이터를 만들거나 대규모의 데이터를 일괄적으로 초기화해야할 경우에 ndarray를 연속값이나 0또는1로 초기화해 쉽게 생성해야 할 필요가 있다.

이 경우 arange( ), zeros( ), ones( )를 이용해 쉽게 ndarray를 생성할 수 있다.

arange( ) : 0부터 함수 인자 값 -1까지의 값을 순차적으로 ndarray의 데이터값으로 변환해 준다.

default 함수 인자는 stop값이어서 0부터 stop값인 10-1인 9까지의 연속 숫자값을 보여준다

range와 유사하게 start값도 부여해 0이 아닌 다른 값부터 시작한 연속 값을 부여할 수도 있다

zeros( ): 함수 인자로 튜플 형태의 shape 값을 입력하면 모든 값을 0으로 채운 해당 shape을 가진 ndarray를 반환한다.

ones( ): 함수 인자로 튜플 형태의 shape 값을 입력하면 모든 값을 1로 채운 해당 shape을 가진 ndarray를 반환한다.

함수 인자로 dtype을 정해주지 않으면 default로 float64형의 데이터로 ndarray를 채운다

ndarray의 차원과 크기를 변경하는 reshape( )

reshape( ) 메서드는 ndarray를 특정 차원 및 크기로 변환한다

변환을 원하는 크기를 함수인자로 부여하면 된다.

넘파이의 ndarray의 데이터 세트 선택하기 - 인덱싱(Indexing)

1. 특정한 데이터만 추출: 원하는 위치의 인덱스 값을 지정하면 해당 위치의 데이터가 반환된다
2. 슬라이싱(Slicing): 연속된 인덱스상의 ndarray를 추출하는 방식이다. ':' 기호 사이에 시작 인덱스와 종료 인덱스를 표시하면 시작 인덱스에서 종료 인덱스-1 위치에 있는 데이터의 ndarray 값을 반환한다.

ex) 1:5라고 하면 인덱스 1과 종료 인덱스 4에 해당하는 ndarray 반환.

3. 팬시 인덱싱(Fancy Indexing): 일정한 인덱싱 집합을 리스트 또는 ndarray 형태로 지정해 해당 위치에 있는 데이터의 ndarray를 반환한다.
4. 불린 인덱싱(Boolean Indexing): 특정 조건에 해당하는지 여부인 True/False 값 인덱싱 집합을 기반으로 True에 해당하는 인덱스 위치에 있는 데이터의 ndarray 값을 반환한다.

단일 값 추출

1개의 데이터값을 선택하려면 ndarray 객체에 해당하는 위치의 인덱스 값을 [ ] 안에 입력하면 된다.

axis 0: 로우 방향의 축

axis 1: 칼럼 방향의 축

즉, [row=0, col=1] 인덱싱은 [axis 0=0, axis 1=1]

2차원은 axis 0, axis 1이고 3차원 ndarray는 axis 0, axis 1, axis 2로 3개의 축을 가진다. (행, 열, 높이로 이해하면 된다)

슬라이싱

':' 기호를 이용해 연속한 데이터를 슬라이싱해서 추출할 수 있다.

(단일 데이터 추출을 제외하고 슬라이싱, 팬시 인덱싱, 불린 인덱싱으로 추출된 데이터 세트는 모두 ndarray 타입니다)

':' 생략

1. ':' 기호 앞에 시작 인덱스를 생략하면 자동으로 맨 처음 인덱스인 0으로 간주한다.
2. ':' 기호 뒤에 종료 인덱스를 생략하면 자동으로 맨 마지막 인덱스인 0으로 간주한다.
3. ':' 기호 앞/뒤에 시작/종료 인덱스를 생략하면 자동으로 맨 처음/마지막 인덱스로 간주한다.

2차원 ndarray에서 뒤에 오는 인덱스를 없애면 1차원 ndarray를 반환한다. ex) array2d[0] 은 첫 번째 로우 ndarray를 반환한다

팬시 인덱싱

리스트나 ndarray로 인덱스 집합을 지정하면 해당 위치의 인덱스에 해당하는 ndarray를 반환하는 인덱싱 방식이다.

불린 인덱싱

조건 필터링과 검색을 동시에 할 수 있으므로 매우 자주 사용된다.

ndarray의 인덱스를 지정하는 [ ] 내에 조건문을 그대로 기재하면 되는 방식으로, for loop/if else 문보다 훨씬 간단하게 구현할 수 있다.

불린 인덱싱이 동작하는 단계

1. array1d > 5와 같이 ndarray의 필터링 조건을 [ ] 안에 기재
2. False 값은 무시하고 True 값에 해당하는 인덱스값만 저장(True 값 자체인 1을 저장하는 게 아니라 True 값을 가진 인덱스를 저장하는 것이다)
3. 저장된 인덱스 데이터 세트로 ndarray 조회

행렬의 정렬 - sort( )와 argsort( )

행렬 정렬

np.sort( ) 와 같이 넘파이에서 sort( )를 호출하는 방식 - 원 행렬은 그대로 유지한 채 원 행렬의 정렬된 행렬을 반환한다

ndarray.sort( ) 와 같이 행렬 자체에서 sort( )를 호출하는 방식 - 원 행렬 자체를 정렬한 형태로 변환하며 반환 값은 None 이다.

np.sort( )나 ndarray.sort( )모두 기본적으로 오름차순으로 행렬 내 원소를 정렬한다.

내림차순으로 정렬하고 싶을 때는 [::-1]을 적용한다. ex) np.sort( )[::-1]

정렬된 행렬의 인덱스를 반환하기

원본 행렬이 정렬되었을 때 기존 원본 행렬의 원소에 대한 인덱스를 필요로 할 때 np.argsort( )를 이용한다.

np.argsort( )는 정렬 행렬의 원본 행렬 인덱스를 ndarray 형으로 반환한다.

이 때도 내림차순으로 정렬 시에 np.argsort( )[::-1] 적용하면 된다

넘파이의 ndarray는 실제 값과 그 값이 뜻하는 메타 데이터를 별도의 ndarray로 각각 가져야 하기 때문에 argsort( )가 매우 활용도가 높다

선형대수 연산 - 행렬 내적과 전치 행렬 구하기

행렬 내적(행렬 곱)

두 행렬 A와 B의 내적은 np.dot( )을 이용해 계산이 가능하다.

전치 행렬

원 행렬에서 행과 열의 위치를 교환한 원소로 구성한 행렬을 그 행렬의 전치 행렬이라고 한다.

이 때 행렬 A의 전치 행렬은 A^T와 같이 표기한다.

넘파이의 transpose( )를 이용해 전치 행렬을 쉽게 구할 수 있다

04 데이터 핸들링 -판다스

판다스는 파이썬에서 데이터 처리를 위해 존재하는 가장 인기 있는 라이브러리로,

행과 열로 이뤄진 2차원 데이터를 효율적으로 가공/처리할 수 있는 다양하고 훌륭한 기능을 제공한다

판다스의 핵심 객체는 DataFrame이다.

(DataFrame: 여러 개의 행과 열로 이뤄진 2차원 데이터를 담는 데이터 구조체이다)

DataFrame을 이해하기 전에 Index와 Series를 이해하는 것도 중요하다.

Index: RDBMS의 PK처럼 개별 데이터를 고유하게 식별하는 Key 값

Series와 DataFrame은 모두 Index를 Key 값으로 가지고 있다.

그러나 Series는 칼럼이 하나 뿐인 데이터 구조체이고, DataFrame은 칼럼이 여러 개인 데이터 구조체이다.

DataFrame과 리스트, 딕셔너리, 넘파이 ndarray 상호 변환

DataFrame ↔ 파이썬의 리스트, 딕셔너리, 넘파이 ndarray 등

넘파이 ndarray, 리스트, 딕셔너리를 DataFrame으로 변환하기

DataFrame은 넘파이 ndarray와 다르게 칼럼명을 가지고 있어, 상대적으로 편하게 데이터 핸들링이 가능하다

일반적으로 DataFrame으로 변환 시 칼럼명을 지정해주고, 지정하지 않으면 자동으로 칼럼명이 할당된다.

판다스 DataFrame 객체의 생성 인자 data는 리스트나 딕셔너리 또는 넘파이 ndarray를 입력받고, 생성 인자 columns는 칼럼명 리스트를 입력받아서 쉽게 DataFrame을 생성할 수 있다

DataFrame은 기본적으로 행과 열을 가지는 2차원 데이터로, 2차원 이하의 데이터들만 DataFrame으로 변환될 수 있다.

딕셔너리를 DataFrame으로 변환 시에는 딕셔너리의 Key는 칼럼명으로, 딕셔너리의 값(Value)은 키에 해당하는 칼럼 데이터로 변환된다.

따라서 키의 경우는 문자열, 값의 경우는 리스트(또는 ndarray) 형태로 딕셔너리를 구성한다

DataFrame을 넘파이 ndarray, 리스트, 딕셔너리로 변환하기

많은 머신러닝 패키지가 기본 데이터 형으로 넘파이 ndarray를 사용하기 때문에 데이터 핸들링은 DataFrame을 이용하더라도 머신러닝 패키지의 입력 인자 등에 적용하기 위해 넘파이 ndarray로 변환하는 경우가 빈번하게 발생한다.

DataFrame → 리스트: values로 얻은 ndarray에 tolist( ) 호출

DataFrame → 딕셔너리: to_dict( ) 메서드에 인자로 'list' 입력

DataFrame의 칼럼 데이터 세트 생성과 수정

Series에 상숫값을 할당하면 Series의 모든 데이터 세트에 일괄적으로 적용된다

DataFrame 데이터 삭제

drop( ) 메서드 이용

drop( ) 메서드의 원형:

DataFrame.drop(labels=None, axis=0, index=None, colums=None, level=None, inplace=False, errors='raise')

axis 값에 따라서 특정 칼럼 또는 특정 행을 드롭한다 (axis 0: 로우 방향 축, axis 1: 칼럼 방향 축)

drop( ) 메서드에 axis=1을 입력하면 칼럼을 드롭

drop( ) 메서드에 axis=0을 입력하면 특정 로우를 드롭

axis를 0으로 지정하면 DataFrame은 자동으로 labels에 오는 값을 인덱스로 간주한다

Inplace=False이면 자기 자신의 DataFrame의 데이터는 삭제하지 않으며, 삭제된 결과 DataFrame을 반환한다. (예제 참고!)

Inplace=True로 설정하면 자신의 DataFrame의 데이터를 삭제한다. 이 경우에는 반환 값을 다시 자신의 DataFrame 객체로 할당하면 안된다

여러 개의 칼럼을 삭제하고 싶으면 리스트 형태로 삭제하고자 하는 칼럼명을 입력해 labels 파라미터로 입력하면 된다.

drop( ) 메서드에서 axis와 inplace 인자를 적용해 DataFrame을 변경하는 방식

- axis: DataFrame의 로우를 삭제할 때는 axis=0, 칼럼을 삭제할 때는 axis=1으로 설정
- 원본 DataFrame은 유지하고 드롭된 DataFrame을 새롭게 객체 변수로 받고 싶으면 inplace=False로 설정(디폴트 값이 False)
- 원본 DataFrame에 드롭된 결과를 적용할 경우에는 inplace=True를 적용
- 원본 DataFrame에서 드롭된 DataFrame을 다시 원본 DataFrame 객체 변수로 할당하면 원본 DataFrame에서 드롭된 결과를 적용할 경우와 같음.

Index 객체

판다스의 Index 객체는 RDBMS의 PK(Primary Key)와 유사하게 DataFrame, Series의 레코드를 고유하게 식별하는 객체이다.

Series 객체는 Index 객체를 포함하지만 Series 객체에 연산 함수를 적용할 때 Index는 연산에서 제외된다

DataFrame 및 Series에 reset_index( ) 메서드를 수행하면 새롭게 인덱스를 연속 숫자 형으로 할당하며 기존 인덱스는 'index'라는 새로운 칼럼명으로 추가한다

데이터 셀렉션 및 필터링

DataFrame의 [ ] 연산자

- 넘파이에서 [ ] 연산자: 행의 위치, 열의 위치, 슬라이싱 범위 등을 지정해 데이터를 가져온다
- DataFrame의 [ ] 연산자: 칼럼명 문자(또는 칼럼명의 리스트 객체) 또는 인덱스로 변환 가능한 표현식

DataFrame 뒤의 [ ] 연산자의 입력 인자가 여러 가지 형태일 때

- DataFrame 바로 뒤의 [ ] 연산자는 넘파이의 [ ]나 Series의 [ ]와 다르다.
- DataFrame 바로 뒤의 [ ] 내 입력값은 칼럼명(또는 칼럼의 리스트)을 지정해 칼럼 지정 연산에 사용하거나 불린 인덱스 용도로만 사용해야 한다
- DataFrame[0:2]와 같은 슬라이싱 연산으로 데이터를 추출하는 방법은 사용하지 않는 게 좋다.

DataFrame iloc( ) 연산자

판다스는 DataFrame의 로우나 칼럼을 지정하여 데이터를 선택할 수 있는 인덱싱 방식으로 iloc[ ]와 loc[ ]을 제공한다

iloc[ ]: 위치 기반 인덱싱 방식

행과 열 위치를, 0을 출발점으로 하는 세로축, 가로축 좌표 정숫값으로 지정

loc[ ]: 명칭(label) 기반 인덱싱 방식

데이터 프레임의 인덱스 값으로 행 위치 / 칼럼의 명칭으로 열 위치를 지정

iloc[ ]은 위치 기반 인덱싱만 허용하기 때문에 행과 열의 좌표 위치에 해당하는 값으로 정숫값 또는 정수형의 슬라이싱, 팬시 리스트 값을 입력해줘야 한다.

iloc[행 위치의 정숫값, 열 위치의 정숫값]같이 명확하게 DataFrame의 행 위치와 열 위치를 좌표 정숫값 형태로 입력하면 해당 위치의 데이터를 가져올 수 있다

그러나 iloc[ ]에 DataFrame의 인덱스 값이나 칼럼명을 입력하면 오류가 발생한다

iloc[ ]는 열 위치에 -1을 입력하여 DataFrame의 가장 마지막 열 데이터를 가져온다.

iloc[:, -1] : 맨 마지막 칼럼의 값, 즉 타깃 값

iloc[:, :-1] : 처음부터 맨 마지막 칼럼을 제외한 모든 칼럼의 값, 즉 피처

DataFrame loc[ ] 연산자

loc[ ]는 명칭(Label) 기반으로 데이터를 추출한다.

행 위치에는 DataFrame 인덱스 값을, 열 위치에는 칼럼명을 입력해서 loc[인덱스값, 칼럼명]와 같은 형식으로 데이터를 추출할 수 있다

일반적으로 슬라이싱을 '시작값:종료 값'과 같이 지정하면 시작값 ~ 종료 값 -1 까지의 범위인데,

loc[ ]에 슬라이싱 기호를 적용하면 종료값-1이 아니라 종료값까지 포함한다.

왜냐하면 명칭은 숫자형이 아니기 때문에 -1을 할 수 없기 때문이다.

[ ], iloc[ ], loc[ ]의 특징과 주의할 점

1. 개별 또는 여러 칼럼 값 전체를 추출하고자 한다면 data_df['Name']과 같이 DataFrame['칼럼명']만으로도 충분하다. 하지만 행과 열을 함께 사용하여 데이터를 추출해야 한다면 iloc[ ]나 loc[ ]를 사용해야 한다
2. DataFrame의 인덱스나 칼럼명으로 데이터에 접근하는 것은 명칭 기반 인덱싱이다. 0부터 시작하는 행, 열의 위치 좌표에만 의존하는 것이 위치 기반 인덱싱이다.
3. iloc[ ]는 위치 기반 인덱싱만 가능하므로 행과 열 위치 값으로 정수형 값을 지정해 원하는 데이터를 반환한다.
4. loc[ ]는 명칭 기반 인덱싱만 가능하므로 행 위치에 DataFrame 인덱스가 오며, 열 위치에는 칼럼명을 지정해 원하는 데이터를 반환한다.
5. 명칭 기반 인덱싱에서 슬라이싱을 '시작점:종료점'으로 지정할 때 시작점에서 종료점을 포함한 위치에 있는 데이터를 반환한다.

불린 인덱싱

명칭이나 위치 지정 인덱싱에서 가져올 값은 주로 로직이나 조건에 의해 계산한 뒤에 행 위치, 열 위치 값으로 입력되는데, 그럴 필요 없이 처음부터 가져올 값을 조건으로 [ ] 내에 입력하면 자동으로 원하는 값을 필터링 한다.

조건 연산자

1. and 조건일 때는 &
2. or 조건일 때는 |

3. Not 조건일 때는 ~

정렬, Aggregation 함수, GroupBy 적용

DataFrame, Series의 정렬 - sort_values()

sort_values( )의 주요 입력 파라미터는 by, ascending, inplace이다.

- by로 특정 칼럼을 입력하면 해당 칼럼으로 정렬을 수행한다. / 여러 개의 칼럼으로 정
- ascending = True : (기본) 오름차순, ascending = False : 내림차순
- inplace=False : (기본) sort_values( )를 호출한 DataFrame은 그대로 유지하면서 정렬된 DataFrame을 결과로 반환한다.

  inplace=True : 호출한 DataFrame의 정렬 결과를 그대로 정렬한다.

Aggregation 함수 적용

DataFrame에서 바로 aggregation을 호출할 경우 모든 칼럼에 해당 aggregation을 적용한다는 차이가 있다.

groupby( ) 적용

DataFrame의 groupby( ) 사용 시 입력 파라미터 by에 칼럼을 입력하면 대상 칼럼으로 groupby 된다.

DataFrame에 groupby( )를 호출하면 DataFrameGroupBy라는 또 다른 형태의 DataFrame을 반환한다.

DataFrame에 groupby( )를 호출해 반환된 결과에 aggregation 함수를 호출하면 groupby( ) 대상 칼럼을 제외한 모든 칼럼에 해당 aggregation 함수를 적용한다

결손 데이터 처리하기

판다스는 결손 데이터(Missing Data)를 처리하는 편리한 API를 제공한다.

결손 데이터는 칼럼에 값이 없는, 즉 NULL인 경우를 의미하며, 이는 넘파이의 NaN으로 표시한다.

NaN 여부를 확인하는 API는 isna( )이며, NaN 값을 다른 값으로 대체하는 API는 fillna( )이다.

isna()로 결손 데이터 여부 확인

isna()는 데이터가 NaN인지 아닌지 알려준다.

DataFrame에 isna()를 수행하면 모든 칼럼의 값이 NaN인지 아닌지 True나 False로 알려준다.

fillna( )로 결손 데이터 대체하기

fillna( )를 이용하면 결손 데이터를 다른 값으로 대체할 수 있다.

fillna( )를 이용해 반환 값을 다시 받거나 inplace = True 파라미터를 fillna( )에 추가해야 실제 데이터 세트 값이 변경된다.

apply lambda 식으로 데이터 가공

lambda는 함수의 선언과 함수 내의 처리를 한 줄의 식으로 쉽게 변환하는 식이다.

lambda 식은 if else를 지원하는데, if 절의 경우 if 식보다 반환 값을 먼저 기술해야한다.

또한 if, else만 지원하고 if, else if, else와 같이 else if는 지원하지 않는다.

else if를 이용하기 위해서는 else 절을 ()로 내포해 ( ) 내에서 다시 if else 적용해 사용한다.