

파이썬 기초~응용

『3과목 :』

데이터 마트와 데이터 전처리

2025.06.23-07.04(10일, 70시간)

Prepared by Daekyeong Kim

Ph.D.

1. Data Mart & Data Preprocessing
2. Data Structures
3. Data Gathering(Collect, Acquisition), Data Ingestion
4. Data Invest & Exploratory Data Analysis, Data Visualization
5. Data Cleansing (정제)
6. Data Integration (통합)
7. Data Reduction (축소)
8. Data Transformation (변환)
9. Feature Engineering & Data Encoding
10. Cross Validation & Data Splitting
11. Data Quality Assessment and Model Performance Evaluation
12. 『3과목』 Self 점검

『 3과목 :』 데이터 마트와 데이터 전처리

- Data Mart & Data Preprocessing
- Data Structures
- Data Gathering(Collect, Acquisition), Data Ingestion
- Data Invest & Exploratory Data Analysis, Data Visualization
- Data Cleansing (정제)
- Data Integration (통합)
- Data Reduction (축소)
- Data Transformation (변환)
- Feature Engineering & Data Encoding
- Cross Validation & Data Splitting
- Data Quality Assessment and Model Performance Evaluation

- 『3과목』 Self 점검



학습목표

- 이 워크샵에서는 데이터 분석을 실행하기 전 데이터의 표현과 Data pre-processing 을 가능케 할 것입니다. Data Scientist의 기본역량인 Data 처리 및 AI 활용 역량을 가질 수 있습니다.

눈높이 체크

- Data pre-processing 을 알고 계신가요?



1. Data pre-processing

Data pre-processing

- Data pre-processing?

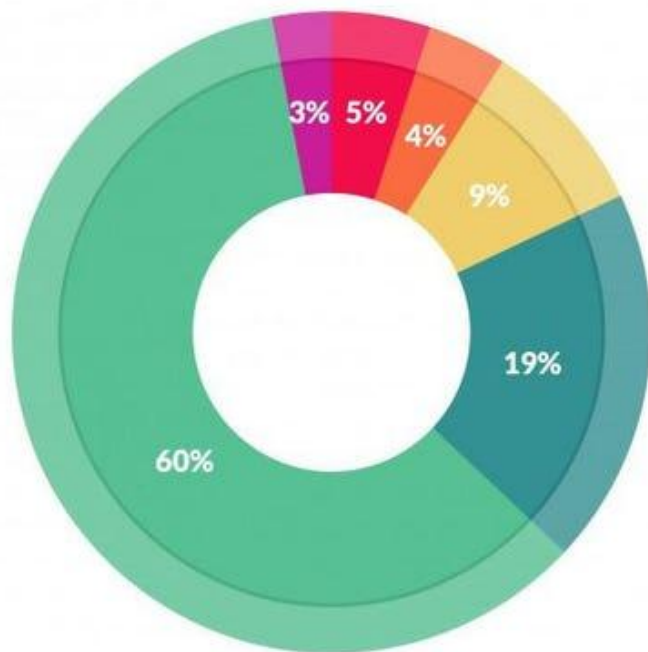
- Data pre-processing에 대한 논의는 플랫폼 제공업체인 CrowdFlower의 약 80명의 데이터 과학자를 대상으로 한 설문 조사에 대한 Gil Press의 "Cleaning Big Data: Most Time-Consuming, Least Enjoyable Data Science Task, Survey Says: Mar 23, 2016"를 통해서이다.
- "데이터 py3_10_basic는 데이터 과학자 작업의 약 80%를 차지합니다. 데이터 과학자는 데이터를 정리하고 구성하는 데 시간의 60%를 사용합니다. 데이터 세트 수집은 시간의 19%로 두 번째입니다. 즉, 데이터 과학자는 분석을 위한 데이터 및 관리에 시간의 약 80%를 소비합니다."



1. Data pre-processing

Data pre-processing

- Data pre-processing?



What data scientists spend the most time doing

- Building training sets: 3%
- Cleaning and organizing data: 60%
- Collecting data sets: 19%
- Mining data for patterns: 9%
- Refining algorithms: 4%
- Other: 5%



1. Data pre-processing

Data pre-processing

- Gil Press는 또한 같은 글에서 “In 2009, data scientist Mike Driscoll popularized the term “data munging,” describing the “painful process of cleaning, parsing, and proofing one’s data” as one of the three Gender skills of data geeks.” 라고 말하며, Data pre-processing에 대한 통찰을 보여 주고 있는 데, 그것을 정리하면 “data munging”이라 하며 그 내용은 다음과 같다.
 - Data cleaning and organizing
 - Data cleaning, parsing, and proofing



1. Data pre-processing

Data pre-processing

- 그렇다면, “data munging”은 무엇인지 계속 살펴보면,
“

Data wrangling, sometimes referred to as data munging, is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics...

The goal of data wrangling is to assure quality and useful data...Data wrangling typically follows a set of general steps which begin with extracting the data in a raw form from the data source, "munging" the raw data (e.g. sorting) or parsing the data into predefined data structures, and finally depositing the resulting content into a data sink for storage and future use.

“



1. Data pre-processing

Data pre-processing

- 데이터 랭글링?

- 데이터 랭글링 data wrangling은 원본 데이터를 정제하고 사용 가능한 형태로 구성하기 위한 변환 과정을 광범위하게 의미하는 비공식적인 용어. 데이터 랭글링은 데이터 전처리의 한 단계에 불과하지만 중요한 단계
- **Gather**
데이터를 얻는 방법으로는 다운로드(Download), 웹 스크레이핑(Web scrapping), API(Application Programming Interface)가 있다.
- **Assess**
얻은 데이터를 읽고 데이터가 깨끗한지 아닌지 판단하는 단계.
- **Clean**
데이터를 정제하는 방법으로는 Define, Code, Test가 있습니다. 2단계(Assess)에서 발견된 데이터의 문제점을 보고 어떤 부분을 정제할지 정의하고(Define), 정제하기 위한 코드를 짜고(Code), 잘 정제가 되었는지 테스트를 해보는(Test) 것.
- **Reassess and Iterate**
다시 2단계로 돌아가 데이터가 잘 정제되었는지 판단을 합니다. 추가로 정제해야 할 부분이 있다면 다시 2-3-4 단계를 반복.
- **Store (optional)**
나중에 다시 사용하기 위해 저장하는 단계.



1. Data pre-processing

Data pre-processing

- 그렇다면, Data pre-processing 에 대한 정의를 좀 더 살펴보자.

“... manipulation or dropping of data... Examples of data preprocessing include cleaning, instance selection, normalization, one hot encoding, transformation, feature extraction and selection, etc. The product of data preprocessing is the final training set.

...

Tasks of data preprocessing

- Data cleansing
- Data editing
- Data reduction
- Data wrangling”



1. Data pre-processing

Data pre-processing

- 다음 예를 보자,

예

설명

		Gender	Pregnant
Adult	1	Male	No
	2	Female	Yes
	3	Male	Yes
	4	Female	No
	5	Male	Yes

- 성별이 남성 또는 여성이고 임신 여부를 가진 5명의 불가능한 데이터 조합의 성인 데이터 셋



1. Data pre-processing

Data pre-processing

- 다음 예를 보자,

Data cleansing

설명

Adult			
		Gender	Pregnant
	1	Male	No
	2	Female	Yes
	4	Female	No

Male	Yes
------	-----

- 데이터세트에 존재하는 이러한 데이터가 사용자 입력 오류 또는 데이터 손상으로 인해 발생한 것으로 판단할 수 있기 때문에 이러한 데이터를 제거.
- 이러한 데이터를 삭제해야 하는 이유는 불가능한 데이터가 데이터 마이닝 프로세스의 후반 단계에서 계산 또는 데이터 조작 프로세스에 영향을 미치기 때문.



1. Data pre-processing

Data pre-processing

- 다음 예를 보자,

Data editing

설명

		Gender	Pregnant
Adult	1	Male	No
	2	Female	Yes
	3	Female	Yes
	4	Female	No
	5	Female	Yes
		Male	Yes

- 성인이 여성이라고 가정하고 그에 따라 변경할 수 있다.
- 데이터 마이닝 프로세스의 후반 단계에서 데이터를 조작을 수행할 때 데이터를 보다 명확하게 분석할 수 있도록 데이터 세트를 편집



1. Data pre-processing

Data pre-processing

- 다음 예를 보자,

Data reduction

설명

Adult			
		Gender	Pregnant
2		Female	Yes
4		Female	No
1		Male	No
3		Male	Yes
5		Male	Yes

- 데이터를 성별로 정렬.
- 이를 통해 데이터 세트를 단순화하고 더 집중하고 싶은 성별을 선택할 수 있다.



1. Data pre-processing

Data pre-processing

● Data pre-processing 에 대한 또 다른 예를 보자,

“

- 데이터 결합 (예: 행 결합, 열 결합, JOIN)
- 데이터 분할, 필터링, 샘플링
- 파생변수 생성 (예: 날짜 → 주말/평일 구분, 점수 → 등급)
- 더미변수 생성 (원-핫 인코딩, 예: 성별 → 0/1)
- 결측치 처리 (제거·보간)
- 이상치 처리 (제거·보간)
- 스케일 조정 (예: MixMax → 0~1, 표준점수, 로그스케일)
- 자료형 변경 (예: String → Datetime, String → Integer)
- 기타 데이터 수정·보정

”



1. Data pre-processing

Data pre-processing

- 마지막으로 NAVER 지식백과를 통해, Data pre-processing 에 대한 정의를 살펴보면

“

원자료(raw data)를 데이터 분석 목적과 방법에 맞는 형태로 처리하기 위하여 불필요한 정보를 분리 제거하고 가공하기 위한 예비적인 조작...데이터의 측정 오류를 줄이고 잡음(noise), 왜곡, 편차를 최소화한다. 정밀도, 정확도, 이상 값(outlier), 결측 값(missing value), 모순, 불일치, 중복 등의 문제를 해결하기 위한 방법으로 다음 전처리 방법들을 사용한다.

- 데이터 정제(cleansing): 결측 값(missing value; 빠진 데이터)들을 채워 넣고, 이상치를 식별 또는 제거하고, 잡음 섞인 데이터를 평활화(smoothing)하여 데이터의 불일치성을 교정하는 기술
- 데이터 변환(transformation): 데이터 유형 변환 등 데이터 분석이 쉬운 형태로 변환하는 기술. 정규화(normalization), 집합화(aggregation), 요약(summarization), 계층 생성 등의 방법을 활용한다.
- 데이터 필터링(filtering): 오류 발견, 보정, 삭제 및 중복성 확인 등의 과정을 통해 데이터의 품질을 향상하는 기술
- 데이터 통합(integration): 데이터 분석이 용이하도록 유사 데이터 및 연계가 필요한 데이터들을 통합하는 기술
- 데이터 축소(reduction): 분석 시간을 단축할 수 있도록 데이터 분석에 활용되지 않는 항목 등을 제거하는 기술

”



1. Data pre-processing

Data pre-processing

- 이상의 내용을 통해, Data pre-processing 작업을 요약해 보면 다음과 같다.

- ① Data cleaning and organizing
- ② Data cleaning, parsing, and proofing
- ③ the raw data (e.g. sorting) or parsing the data into predefined data structures, and finally depositing the resulting content into a data sink for storage and future use
- ④ Data cleansing
- ⑤ Data editing
- ⑥ Data reduction
- ⑦ Data wrangling
- ⑧ 데이터 정제(cleansing)
- ⑨ 데이터 변환(transformation)
- ⑩ 데이터 필터링(filtering)
- ⑪ 데이터 통합(integration)
- ⑫ 데이터 축소(reduction)

- ① 데이터 결합 (예: 행 결합, 열 결합, JOIN)
- ② 데이터 분할, 필터링, 샘플링
- ③ 파생변수 생성 (예: 날짜 → 주말/평일 구분, 점수 → 등급)
- ④ 더미변수 생성 (원-핫 인코딩, 예: 성별 → 0/1)
- ⑤ 결측치 처리 (제거·보간)
- ⑥ 이상치 처리 (제거·보간)
- ⑦ 스케일 조정 (예: MixMax → 0~1, 표준점수, 로그스케일)
- ⑧ 자료형 변경 (예: String → Datetime, String → Integer)
- ⑨ 기타 데이터 수정·보정

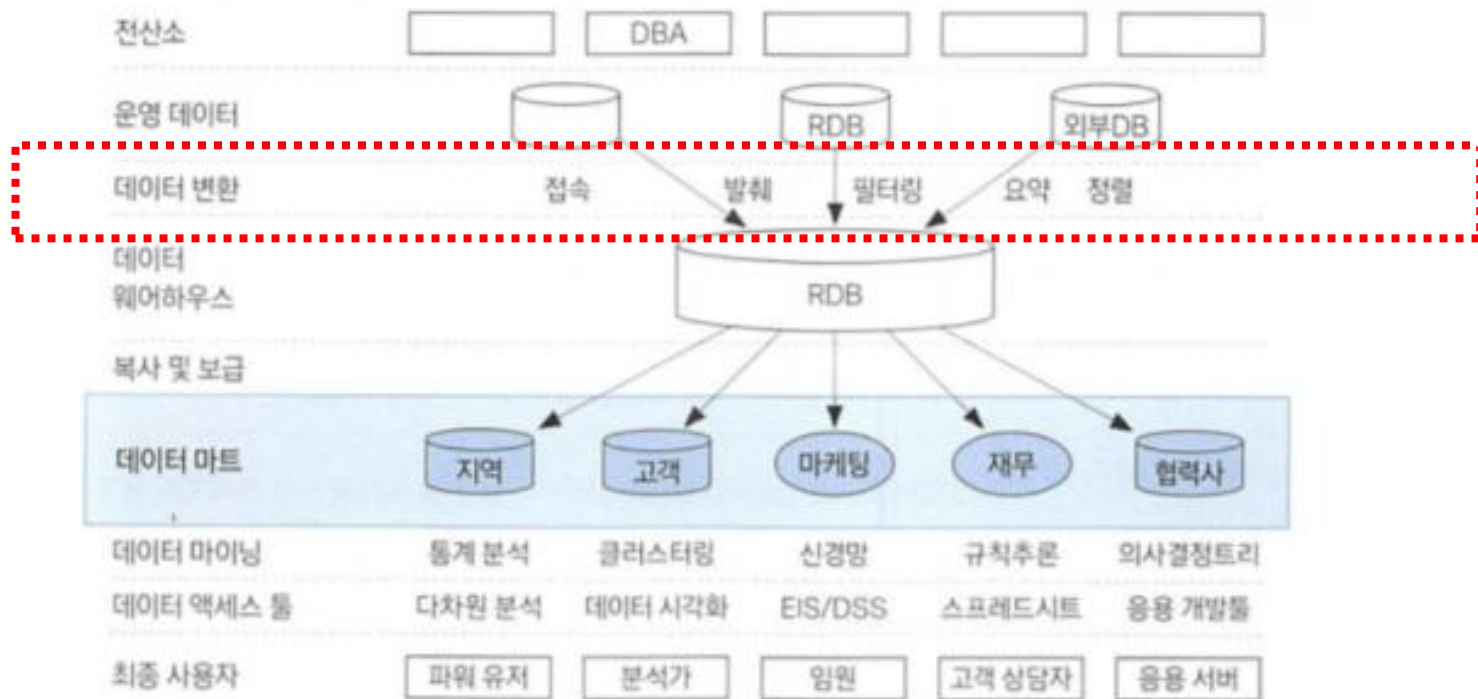


1. Data Preprocessing

데이터 마트 개발

● 데이터마트

- 데이터 웨어하우스와 사용자의 중간층에 위치한 것. 하나의 주제 또는 하나의 부서 중심의 데이터 웨어하우스 라고 할 수 있다.
- 데이터마트 내 대부분 데이터는 데이터 웨어하우스로부터 복제되지만, 자체적으로 수집될 수 있으며, 관계형 데이터베이스나 다차원 데이터베이스를 이용하여 구축한다.





1. Data Preprocessing

데이터 마트 개발

- 데이터 전처리(Data Preprocessing)와 데이터 웨어하우스의 관계:
 - 데이터 전처리는 데이터 분석을 위해 데이터를 py3_10_basic하고 정리하는 과정으로, 데이터 웨어하우스에 저장되는 데이터의 일관성과 품질을 유지하는 데 중요한 역할을 합니다.
 - 데이터 전처리는 데이터의 클린징, 변환, 통합, 축소 등의 과정을 포함하여 데이터를 분석 가능한 형태로 만듭니다.
 - 데이터 전처리는 데이터 웨어하우스에 저장된 데이터를 정제하고 가공하여, 비즈니스 분석 및 의사 결정에 활용할 수 있는 형태로 만듭니다.
- 따라서 데이터 웨어하우스에 저장된 데이터는 데이터 전처리를 거치고 정제되어 분석가능한 상태로 유지됩니다.



1. Data Preprocessing

데이터 마트 개발

● 데이터 획득(Data Acquisition):

- 데이터 획득은 데이터를 얻는 과정을 말합니다.
- 외부 소스 또는 다양한 시스템으로부터 데이터를 수집하거나 가져오는 단계입니다.
- 데이터 획득 단계에서는 데이터를 어디서 가져올 것인지 결정하고, 필요한 데이터를 식별하여 수집합니다.
- 데이터를 수집하는 방법에는 API 호출, 데이터베이스 쿼리, 웹 스크래핑, 외부 소스로부터의 파일 다운로드 등이 포함될 수 있습니다.
- 데이터 획득 단계는 분석이나 의사 결정을 위해 필요한 데이터를 확보하는 데 중요합니다.



1. Data Preprocessing

데이터 마트 개발

● 데이터 주입(Data Ingestion):

- 데이터 수집은 데이터를 수집하여 저장소 또는 데이터 웨어하우스 등으로 이동하는 프로세스입니다.
- 데이터를 가져와서 처리할 수 있는 형태로 변환하고, 저장소에 저장하여 이후 분석 및 활용을 위해 `py3_10_basic`합니다.
- 이 단계에서 데이터를 정제하고 변환하여 일관된 형식으로 저장하며, 데이터 웨어하우스나 데이터마트 등에 데이터를 적재(load)합니다.
- 대용량 데이터를 다루는 경우 데이터 수집 단계는 데이터의 효율적인 저장 및 처리를 위해 중요한 역할을 합니다.



1. Data Preprocessing

데이터 마트 개발

- 데이터 조사(Data Investigation):
 - 데이터 조사는 수집된 데이터를 분석하고 이해하기 위한 과정을 말합니다.
 - 데이터의 내용, 구조, 품질 등을 살펴보고 데이터의 패턴이나 특성을 파악하는 단계입니다.
 - 이 단계에서는 데이터 시각화, 요약 통계, 탐색적 데이터 분석(EDA, Exploratory Data Analysis) 등을 통해 데이터를 탐색합니다.
 - 데이터 조사를 통해 데이터의 유효성을 검증하고, 분석 목적에 맞는 데이터 처리 및 모델링을 위한 기반을 마련합니다.



1. Data Preprocessing

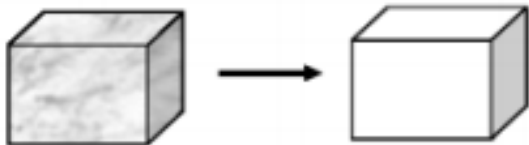
데이터 마트 개발

- 데이터 전처리(Data Preprocessing)와 데이터 웨어하우스의 관계:
 - 데이터 전처리는 데이터 분석을 위해 데이터를 py3_10_basic하고 정리하는 과정으로, 데이터 웨어하우스에 저장되는 데이터의 일관성과 품질을 유지하는 데 중요한 역할을 합니다.
 - 데이터 전처리는 데이터의 클린징, 변환, 통합, 축소 등의 과정을 포함하여 데이터를 분석 가능한 형태로 만듭니다.
 - 데이터 전처리는 데이터 웨어하우스에 저장된 데이터를 정제하고 가공하여, 비즈니스 분석 및 의사 결정에 활용할 수 있는 형태로 만듭니다.
- 따라서 데이터 웨어하우스에 저장된 데이터는 데이터 전처리를 거치고 정제되어 분석가능한 상태로 유지됩니다.

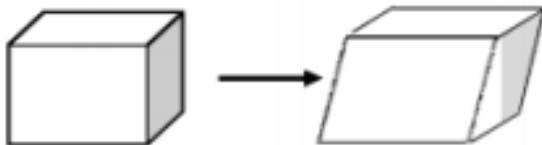
1. Data Preprocessing

데이터 마트 개발

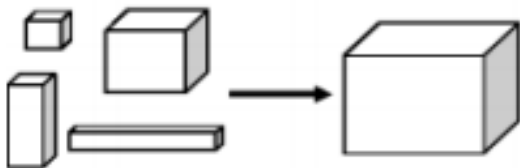
Data cleaning



Data transformation



Data integration

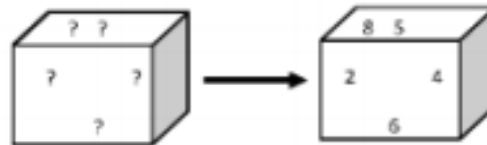


Data Reduction

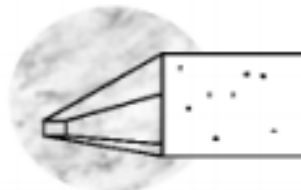
Data normalization



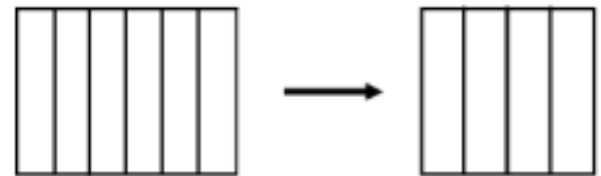
Missing values imputation



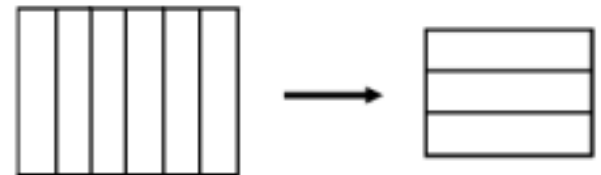
Noise identification



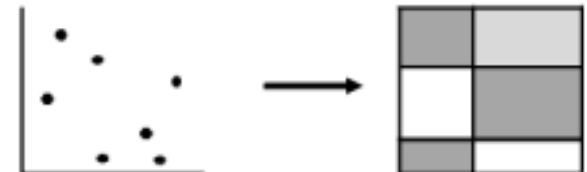
Feature selection



Instance selection



Discretization





1. Data Preprocessing

데이터 마트 개발

- 클린징 (Cleansing):

- 클린징은 데이터에서 불필요하거나 오류가 있는 부분을 찾아내고 수정 또는 제거하는 과정을 말합니다.
- 이상치나 결측치와 같은 오류를 수정하거나 제거하여 데이터의 품질을 향상시킵니다.
- 예를 들어, 결측 데이터를 대체하거나 제거하고, 이상치를 식별하여 수정하는 작업 등이 이 단계에서 이루어집니다.

- 통합 (Integration):

- 통합은 여러 소스에서 추출된 데이터를 하나의 일관된 데이터로 통합하는 과정입니다.
- 데이터를 통합하여 중복을 제거하고, 서로 다른 데이터 소스의 데이터를 일관된 형태로 통일시킵니다.
- 예를 들어, 서로 다른 데이터베이스에서 추출한 고객 정보를 하나의 테이블로 통합하거나, 데이터 형식을 체계화하는 등의 작업이 이루어집니다.



1. Data Preprocessing

데이터 마트 개발

● 축소 (Reduction):

- 축소는 데이터의 크기를 줄이거나 요약하여 저장 및 처리의 효율성을 높이는 과정입니다.
- 불필요한 데이터를 제거하거나, 데이터의 차원을 축소하거나 요약하여 필요한 정보를 보다 간결하게 표현합니다.
- 예를 들어, 차원 축소 기법을 사용하여 변수를 선택하거나, 데이터 압축 기법을 적용하여 데이터 크기를 줄이는 등의 작업을 포함합니다.

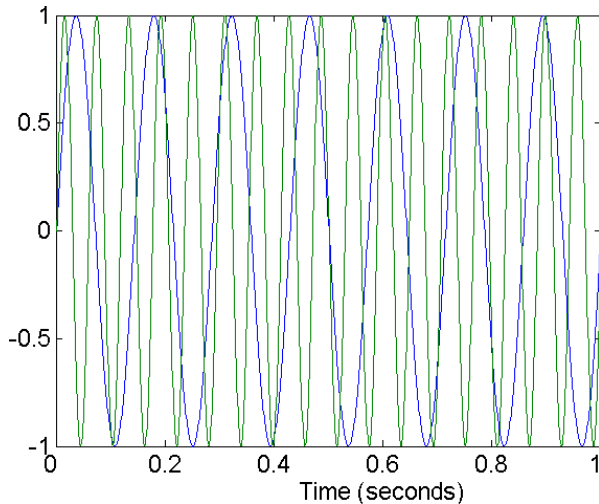
● 변환 (Transformation):

- 변환은 데이터를 분석에 적합한 형태로 변형하는 과정입니다.
- 데이터의 스케일링, 정규화, 표준화, 이산화, 인코딩 등과 같은 변형 작업을 수행합니다.
- 예를 들어, 연속형 데이터를 특정 범위로 스케일링하거나, 범주형 데이터를 숫자로 인코딩하는 등의 변형을 포함합니다.

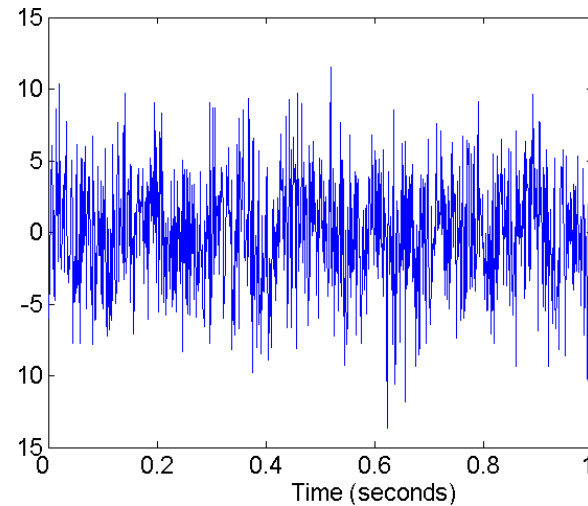
2. 데이터 품질 Data Quality

잡음 Noise

- 측정 과정에서 무작위로 발생하여 측정값의 에러를 발생시키는 것
 - 실제 데이터는 매끈한 곡선 형태의 시계열 데이터였지만 측정 과정에서 잡음이 포함됨 으로 인해 실제 값과 다른 데이터를 얻게 되어 실제 데이터의 형태를 잃어버릴 수도 있음



Two Sine Waves



Two Sine Waves + Noise

2. 데이터 품질 Data Quality

아티팩트 Artifact

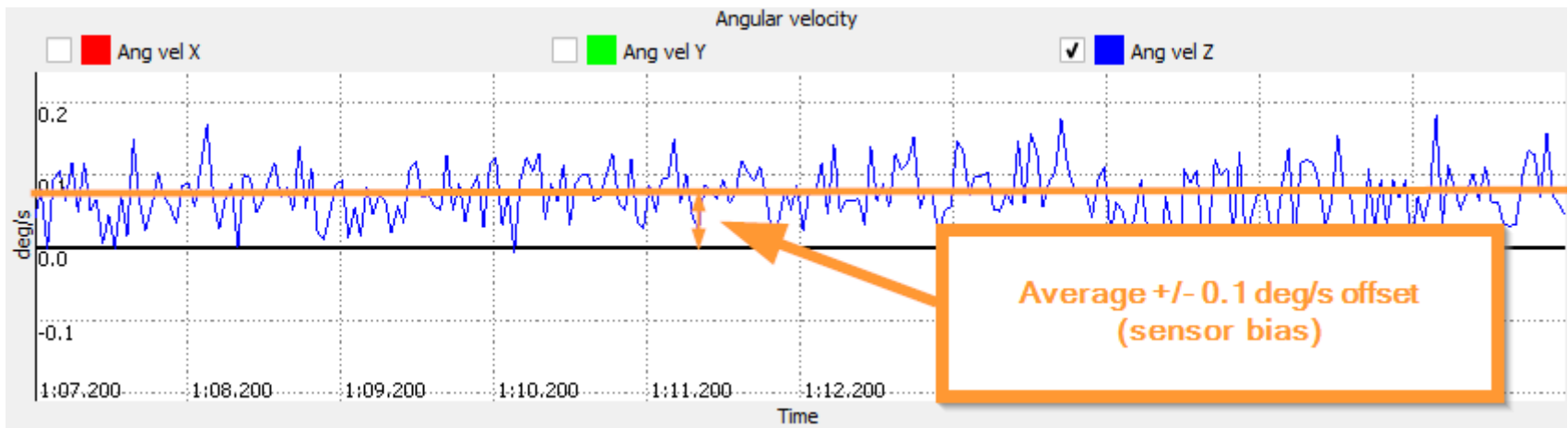
- 어떠한 요인으로 인해 반복적으로 발생하는 왜곡이나 에러를 의미
 - 일례로 카메라를 이용한 영상 데이터 획득에 있어 카메라 렌즈에 얼룩이 묻어 있다면 이 에 해당하는 부분에서는 이 얼룩으로 인한 왜곡이 지속적으로 발생



2. 데이터 품질 Data Quality

바이어스 Bias

- 측정 장비에 포함된 시스템적인 변동으로 앞서 영점 조절 되지 않은 체 중계가 좋은 예
 - 정밀도에서 언급된 예제의 경우 1g에 대한 측정 평균은 1.001이며 이 측정 장비에는 0.001
 - 만큼의 바이어스가 포함되어 있음을 알 수 있음





2. 데이터 품질 Data Quality

정확도 Accuracy

- 정확도는 정확도와 바이어스에 기인하는 것이지만 이를 이용하여 명시적으로 나타낼 수 있는 수식은 없음
 - 다만 정확도는 유효 숫자 Significant digit의 사용에 있어 중요한 측면을 가지고 있음
 - 이는 공학이나 과학에서 기본적으로 다루는 개념으로 수의 정확도에 영향을 주는 숫자를 의미
- 예를 들어, 측정에 있어 이는 측정 장비의 한계로 인해 정확하지 않은 자리의 수를 측정 함에 따라 발생할 수 있는 문제로 자를 이용한 길이 측정을 가정
 - 자의 최소 눈금이 1mm라면, 1mm단위로 길이를 측정하게 될 것이며 이 경우 항상 $\pm 0.5\text{mm}$ 만큼의 오차를 가지게 됨
 - 이 자를 이용하여 측정한 길이가 10.3mm였다면 1mm미만의 값인 0.3mm라는 수치는 의미 가 없음을 알 수 있음



2. 데이터 품질 Data Quality

이상치 Outlier

- 대부분의 데이터와 다른 특성을 보이거나 특정 속성의 값이 다른 개체들과 달리 유별난 값을 가지는 데이터를 의미
 - 이상치의 중요한 점은 잡음과는 다르다는 것
 - 잡음이 임의로 발생하는 예측하기 어려운 요인임에 반해 이상치는 적법한 하나의 데이터로서 그 자체가 중요한 분석의 목적이 될 수도 있음
- 예를 들어 네트워크의 침입자 감시와 같은 응용에 있어서는 대다수의 일반 접속 중 예외적으로 발생하는 불법적인 접속 시도와 같은 이상치를 찾는 것이 주된 목표



2. 데이터 품질 Data Quality

결측치 Missing values

- 데이터의 결측은 일반적인 경우는 아니지만 드물게 발생하는 문제
 - 설문조사의 경우 몇몇 사람들은 자신의 나이나 몸무게와 같은 사적인 정보를 공개하는 것을 꺼리는 경우가 발생하며 이러한 값들은 조사에 있어 결측값으로 남게 됨



2. 데이터 품질 Data Quality

모순, 불일치 Inconsistent values

- 때에 따라서는 동일한 개체에 대한 측정 데이터가 다르게 나타나는 경우가 발생할 수 있는데 이러한 경우를 모순 또는 불일치값이라 표현
 - 예를 들어, 고객의 주소와 우편번호를 저장해 놓은 데이터를 생각해보면, 주소가 동일한 지역임에도 불구하고 어떠한 이유로 우편번호가 상이한 경우가 발생할 수 있음
- 이런 경우에는 주소를 확인해서 우편번호를 정정하는 작업이 필요



2. 데이터 품질 Data Quality

중복 Duplicate data

- 데이터의 중복은 언제든지 발생 가능
 - 문제는 중복된 데이터 사이에 속성의 차 이나 값의 불일치가 발생할 수 있다는 것
- 기본적으로 모든 속성 및 값이 동일하다 면 하나의 데이터는 삭제할 수 있지만, 그 령지 않은 경우에는 두 개체를 합쳐서 하 나의 개체를 만들거나, 응용에 적합한 속 성을 가진 데이터를 선택하는 등의 추가 적인 작업을 필요로 하게 됨



2. 데이터 품질 Data Quality

정밀도 Precision

- 동일한 대상을 반복적으로 측정하였을 때 의 각 결과의 친밀성을 나타내는 것
- 측정 결과의 표준편차 standard deviation로 나타낼 수도 있음
 - 예를 들어 동일한 1g을 측정하는데 있어 각각의 측정 결과가 {1.015, 0.990, 1.013, 1.001, 0.986}인 경우 이들의 표준편차는 0.013이므로 이 때의 정밀도는 0.013이라 말할 수 있음

3 데이터 전처리

데이터 마트 및 데이터 전처리 단계 (Data Preprocessing Pipeline)

1. Data Structures

- 데이터의 형태(정형, 반정형, 비정형), 변수의 타입(수치형, 범주형 등)

2. Data Gathering(Collect, Acquisition), Data Ingestion

- 데이터 수집(Collect) / 데이터 획득(Acquisition) / 데이터 유입(Ingestion)

3. Data Invest & Exploratory Data Analysis, Data Visualization

- 통계 요약, 시각화, 패턴 탐색
- ※ "Data Invest" → 일반적으로는 EDA 또는 Data Understanding으로 표현

4. Data Cleansing (정제)

- 결측치 처리, 이상치 제거, 오타 수정, 잡음 제거 등

5. Data Integration (통합)

- 여러 소스/테이블/파일 병합, 공통 키 기반 조인 등

3 데이터 전처리

데이터 마트 및 데이터 전처리 단계 (Data Preprocessing Pipeline)

6. Data Reduction (축소)

- 특성 선택, 차원 축소(PCA 등), 샘플링, 중복 제거

7. Data Transformation (변환)

- 정규화, 표준화, 로그 변환, 이산화 등

8. Feature Engineering & Data Encoding

- 새로운 변수 생성, 파생변수, 원-핫 인코딩, 라벨 인코딩 등

9. Data Splitting

- 학습 / 검증 / 테스트 세트로 나누기

10. Data Quality Assessment and Model Performance Evaluation

- 전처리 결과 점검, 데이터 분포 확인, 모델 입력 적합성 평가 등
- ※ "데이터 성능 측정" → 일반적으로는 전처리 품질 확인 또는 학습 후 모델 성능 평가 단계에서 수행

『 3과목 :』 데이터 마트와 데이터 전처리

- Data Mart & Data Preprocessing
- Data Structures
- Data Gathering(Collect, Acquisition), Data Ingestion
- Data Invest & Exploratory Data Analysis, Data Visualization
- Data Cleansing (정제)
- Data Integration (통합)
- Data Reduction (축소)
- Data Transformation (변환)
- Feature Engineering & Data Encoding
- Cross Validation & Data Splitting
- Data Quality Assessment and Model Performance Evaluation

『3과목』 Self 점검



학습목표

- 이 워크샵에서는 Pandas의 Series와 DataFrame을 사용할 수 있다.

눈높이 체크

- Pandas를 알고 계신가요?
- Series와 DataFrame을 알고 계신가요?



1. Pandas

라이브러리 py3_10_basic

- 데이터 처리와 분석을 위한 파이썬 라이브러리.
 - R의 data.frame을 본떠서 설계한 DataFrame이라는 데이터 구조를 기반으로 만들어졌다.
 - NumPy가 파이썬에서 수치연산을 잘 할 수 있도록 지원하는 라이브러리인데 이를 기반으로 tabular 데이터(표 형태의 정형데이터)를 보다 편하게 다룰 수 있도록 하는 라이브러리가 Pandas이다. 특히 Pandas를 사용하는 시점부터 파일 입출력(read, write)이 매우 편해지고 데이터 필터링, 그래프, 결측치 처리, 시계열 분석, 스타일링 등 슬슬 엑셀을 대체할 수 있는 기능을 맛보게 된다.
 - 다음 명령으로 설치한다.

```
pip install pandas
```

- 임포트할 때는 보통 pd라는 별명으로 임포트한다.

```
import pandas as pd
```

```
pd.__version__
```




1. Pandas

Data Structures

- 수집, 저장, 데이터 셋 적재 Gathering Data 후 DataFrame에 담을 수 있어야 한다. 대부분의 데이터는 시계열(series)이나 표(table)의 형태로 나타낼 수 있다.
- 데이터 랭글링에 사용되는 가장 일반적인 데이터 구조는 데이터프레임이다. 사용하기 쉽고 기능이 많음. 판다스(Pandas) 패키지는 이러한 데이터를 다루기 위한 시리즈(Series) 클래스와 데이터프레임(DataFrame) 클래스를 제공한다.
- 데이터프레임은 표 형식 데이터로서 PANDAS를 통해 사용할 수 있다.



1. Pandas

Data Structures

- PANDAS를 이해하기위한 핵심 중 하나는 데이터 모델을 이해하는 것입니다. Pandas의 핵심에는 세 가지 데이터 구조가 있다.

DATA STRUCTURE	DIMENSIONALITY	SPREADSHEET ANALOG
Series	1D	Column
DataFrame	2D	Single Sheet
Panel	3D	Multiple Sheets

- 가장 널리 사용되는 데이터 구조는 각각 배열 데이터와 테이블 형식 데이터를 처리하는 Series 및 DataFrame. 스프레드 시트 세계와의 비유는 이러한 유형 간의 기본적인 차이점을 보여준다. DataFrame은 행과 열이 있는 시트와 비슷하지만 Series는 데이터의 단일 열과 비슷합니다. 패널은 시트 그룹. 마찬가지로, 팬더에서 패널은 여러 데이터 프레임을 가질 수 있으며, 각 프레임은 차례로 여러 시리즈를 가질 수 있다.

2. Series

개요

- Pandas 라이브러리의 기본 객체이자 1차원 객체인 시리즈 (Series) 객체는 1차원 배열과 유사하지만, 각각의 원소에 라벨을 붙일 수 있다. 이 라벨을 통해 인덱싱(indexing)이 가능하며, 인덱싱을 통해 원소에 접근할 수 있다. 그리고 NumPy의 어레이 객체보다 강력하고 다양한 메서드를 지원하며 심지어 간단한 그래프도 쉽게 그릴 수 있다.
- 시리즈는 인덱스와 값으로 이루어져 있으며 각 요소는 `.index`와 `.values` 어트리뷰트로 접근할 수 있다. 특징
 - 레이블 또는 데이터의 위치를 지정한 추출 가능.
 - 산술 연산 가능.

ARTIST DATA	
0	145
1	142
2	38
3	13

Index → value

2. Series

생성

- 시리즈 객체의 생성은 Series() 함수에 주로 리스트나 NumPy 어레이 객체를 입력으로 넣어 할 수 있으며 필요시 인덱스를 각 원소의 개수만큼 할당할 수 있다.
- 비어있는 시리즈 객체를 생성할 경우 함수 내부에 아무것도 쓰지 않는다.

파일

소스코드

py3_10_basic

실습환경

```
import pandas as pd  
import numpy as np
```

소스코드

```
pd.Series()
```

결과값1

```
Series([], dtype: object)
```

비고

2. Series

생성

- 리스트와 NumPy 어레이를 사용하여 시리즈 객체를 생성하는 예제이다. 리스트도 중첩이 아니고 어레이도 1차원을 입력으로 하는 것을 알 수 있다. 기본 객체나 NumPy 객체와는 다르게 인덱스가 직접적으로 표기된 것을 알 수 있다. 이것이 Pandas 객체가 다른 객체와 차별화 되는 요소라고 할 수 있겠다. Pandas 객체의 경우 인덱스를 이렇게 확인할 수 있고 원하는 인덱스로 바꿀 수도 있으며 인덱스기반의 다양한 연산 또한 가능하다.

파일	소스코드
실습환경	py3_10_basic
소스코드	<pre>pd.Series([100, 200, 300]) pd.Series(np.array([100, 200, 300]))</pre>
결과값1	<pre>0 100 1 200 2 300 dtype: int64</pre>
비고	

2. Series

생성

- 인덱스를 지정하는 시리즈 객체 생성은 다음과 같다.

파일	소스코드
실습환경	py3_10_basic
소스코드	<pre>pd.Series([100, 200, 300], index = ["A", "B", "C"])</pre>
결과값1	<pre>A 100 B 200 C 300 dtype: int64</pre>
비고	



2. Series

생성

- 인덱스를 지정하여 시리즈를 생성하고자 한다면 딕셔너리 객체를 사용할수도 있다.

파일	소스코드
실습환경	py3_10_basic
소스코드	<pre>pd.Series({"A": 100, "B": 200, "C": 300})</pre>
결과값1	<pre>A 100 B 200 C 300 dtype: int64</pre>
비고	

2. Series

Series CRUD

- 시리즈 객체의 인덱싱은 크게 인덱서(indexer)를 사용하는 것과 사용하지 않는 것이 있다. 색인은 목록을 사용하여 두 번째 매개 변수로 지정.

파일	소스코드
실습환경	py3_10_basic
소스코드	<pre>george_dupe = pd.Series([10, 7, 1, 22],index=['1968', '1969', '1970', '1971'],name='George Songs') george_dupe</pre>
결과값1	<pre>1968 10 1969 7 1970 1 1971 22 Name: George Songs, dtype: int64</pre>
비고	george_dupe라는 이름의 시리즈가 생성되고, 각 연도를 인덱스로 하고 해당 연도의 George의 노래 수를 값으로 가지게 됩니다.

2. Series

Series CRUD

● Series CRUD

파일	소스코드
실습환경	py3_10_basic
소스코드	<pre>george_dupe.index george_dupe.values george_dupe.value_counts george_dupe.value_counts()</pre>
결과값1	<pre>Index(['1968', '1969', '1970', '1971'], dtype='object') array([10, 7, 1, 22], dtype=int64) <bound method IndexOpsMixin.value_counts of 1968 10 1969 7 1970 1 1971 22 Name: George Songs, dtype: int64> George Songs 10 1 7 1 1 1 22 1 Name: count, dtype: int64</pre>



2. Series

Series CRUD

- Series CRUD

파일

소스코드

`george_dupe.index`: 이 코드는 `george_dupe` 시리즈의 인덱스를 반환합니다. 인덱스는 시리즈의 각 요소를 식별하는 데 사용되는 레이블입니다. 여기서는 1968년부터 1971년까지의 연도가 인덱스로 사용되었습니다.

`george_dupe.values`: 이 코드는 `george_dupe` 시리즈의 값만을 반환합니다. 여기서는 George의 각 연도별 노래 수가 값으로 사용되었습니다.

비고

`george_dupe.value_counts`: 이 코드는 시리즈의 값(value)의 빈도수를 반환합니다.

시리즈에 중복된 값이 없기 때문에 모든 값의 빈도수는 1로 동일할 것입니다. 그러므로 `george_dupe.value_counts()`를 호출하면 각 값에 대한 빈도수가 반환됩니다. 시리즈에 중복된 값이 없기 때문에 모든 값의 빈도수는 1로 동일할 것입니다. 그러므로 `george_dupe.value_counts()`를 호출하면 각 값에 대한 빈도수가 반환됩니다.



2. Series

Series CRUD

● Reading

파일	소스코드
실습환경	<pre>py3_10_basic george_dupe['1968']</pre>
소스코드	<pre>george_dupe != 1 george_dupe.loc[george_dupe != 1] 10</pre>
결과값1	<pre>1968 True 1969 True 1970 False 1971 True Name: George Songs, dtype: bool 1968 10 1969 7 1971 22 Name: George Songs, dtype: int64</pre>
비고	<p>loc 메서드는 레이블을 기반으로 특정 항목을 선택하는 데 사용됩니다. loc 메서드는 레이블을 기반으로 특정 항목을 선택하는 데 사용됩니다.</p>



2. Series

Series CRUD

● Updating

파일	소스코드
실습환경	py3_10_basic
소스코드	<pre>george_dupe['1969'] = 6 george_dupe['1969']</pre>
결과값1	6
비고	

● Deletion

파일	소스코드
실습환경	py3_10_basic
소스코드	<pre>del george_dupe['1971'] george_dupe</pre>
결과값1	<pre>1968 10 1969 6 1970 1 Name: George Songs, dtype: int64</pre>
비고	

인덱싱

- 인덱서는 순수하게 정수만 사용가능한 `.iloc[]`와 다양한 입력을 받는 `.loc[]` 두 가지가 있다.

METHOD	WHEN TO USE
속성 액세스	이름이 유효한 경우 단일 색인 이름에 대한 값 가져 오기 속성 이름.
인덱스 액세스	이름이 아닌 경우 단일 인덱스 이름에 대한 값 가져 오기 / 설정 유효한 속성 이름.
<code>.iloc</code>	인덱스 위치 또는 위치별로 값 가져 오기 / 설정. (반 개방 슬라이스 간격)
<code>.loc</code>	색인 레이블로 값 가져 오기 / 설정. (슬라이스 폐쇄 간격)
<code>.iat</code>	인덱스 위치별로 numpy 배열 결과 빠르게 가져 오기 / 설정.
<code>.at</code>	인덱스 레이블로 numpy 배열 결과 빠르게 가져 오기 / 설정

- `loc` 메소드는 특정 범위의 데이터를 인덱싱하는데 사용할 수 있다. 반면 `at` 메소드는 딱 하나의 데이터를 인덱싱할 때 사용한다. `i`는 `integer`의 첫글자를 따서 붙여진 것이다. 인덱스와 컬럼명을 사용하지 않고 대신 행과 컬럼의 위치에 대해서 정수를 사용해 빠른 접근을 하고 싶으면 `iloc`과 `iat`을 사용하면 된다.

인덱싱

● Indexing Summary

파일	소스코드			
실습환경	py3_10_basic			
소스코드	import pandas as pd			
	df = pd.DataFrame([[95, 92, 88], [84, 67, 88], [91, 99, 68], [87, 79, 81], [77, 92, 85]], index=['A', 'B', 'C', 'D', 'E'], columns=['math', 'english', 'history'],) df			
결과값1		math	english	history
	A	95	92	88
	B	84	67	88
	C	91	99	68
	D	87	79	81
	E	77	92	85
비고				

인덱싱

● Indexing Summary

파일	소스코드
실습환경	<pre>py3_10_basic import time # 코드 실행 전 시간 기록 start_time = time.time() # 실행할 코드 작성 print(df.loc[['A', 'B', 'C'], 'english']) print(df.iloc[1:3, 0:2])</pre>
소스코드	<pre># 코드 실행 후 시간 기록 end_time = time.time() # 실행 시간 계산 execution_time = end_time - start_time # 실행 시간 출력 print("코드 실행 시간:", execution_time, "초")</pre>
결과값1	<pre>A 92 B 67 C 99 Name: english, dtype: int64 math english B 84 67 C 91 99 코드 실행 시간: 0.0035009384155273438 초</pre>
비고	

인덱싱

● Indexing Summary

파일	소스코드
실습환경	<pre>py3_10_basic import time # 코드 실행 전 시간 기록 start_time = time.time() # 실행할 코드 작성 print(df.at['C', 'english']) print(df.iat[4, 0])</pre>
소스코드	<pre># 코드 실행 후 시간 기록 end_time = time.time() # 실행 시간 계산 execution_time = end_time - start_time # 실행 시간 출력 print("코드 실행 시간:", execution_time, "초")</pre>
결과값1	<pre>99 77 코드 실행 시간: 0.0009882450103759766 초</pre>
비고	

인덱싱

● Indexing Summary

파일	소스코드
실습환경	<pre>py3_10_basic print(df.loc[['A', 'B', 'C'], ['english', 'history']])</pre>
소스코드	<pre>print(df.at['C', 'english']) print(df.iloc[1:3, 0:2]) print(df.iat[4, 0])</pre>
결과값1	<pre> english history A 92 88 B 67 88 C 99 68 99 math english B 84 67 C 91 99 77</pre>
비고	

Question 1

다음 요구대로 시리즈 요소에 접근해 보시오.

```
george_dupe = pd.Series([10, 7, 1, 22],index=['1968', '1969', '1970', '1971'],name='George Songs')
```

```
george_dupe
```

```
>>
```

```
1968    10
```

```
1969     7
```

```
1970     1
```

```
1971    22
```

```
Name: George Songs, dtype: int64
```

[요구사항]

1. 첫 번째 요소에 접근합니다. 결과는 10.
2. loc를 사용하여 색인 이름을 기반으로 "1968" 연도의 요소에 접근. 결과는 10.
3. iloc를 사용하여 위치 기반으로 0부터 2까지(3미만)의 범위에 해당하는 요소들을 가져옵니다.
4. loc를 사용하여 색인 이름으로 "1968"부터 "1970"까지의 범위에 해당하는 요소들을 가져옵니다.



2. Series

Answer 1

```
print(george_dupe.iloc[0])

print(george_dupe.loc["1968"])

print(george_dupe.iloc[0:3])

print(george_dupe.loc["1968":"1970"])
>>
10

10

1968    10
1969     6
1970     1
Name: George Songs, dtype: int64

1968    10
1969     7
1970     1
Name: George Songs, dtype: int64
```

인덱싱

- `george_dupe.at['1970']`와 `george_dupe.loc['1970']` 모두 Pandas Series에서 특정 레이블(인덱스)에 해당하는 값을 가져오는데 사용됩니다. 하지만 두 방법에는 약간의 차이가 있습니다.

```
george_dupe.at['1970']
```

- 위 코드는 `at` 메서드를 사용하여 레이블(인덱스)이 "1970"인 요소에 접근합니다. 따라서 결과는 해당 요소의 값인 1이 됩니다.

```
george_dupe.loc['1970']
```

- 이 코드는 `loc` 메서드를 사용하여 레이블(인덱스)이 "1970"인 요소에 접근합니다. `loc` 메서드는 슬라이싱과 유사한 방식으로 레이블을 기반으로 데이터를 가져옵니다. 여기서는 "1970"이라는 레이블을 가진 요소를 반환하며, 결과는 1이 됩니다.
- 따라서 `george_dupe.at['1970']`와 `george_dupe.loc['1970']` 모두 같은 값을 반환하며, 이 경우에는 시리즈에서 '1970'이라는 인덱스에 해당하는 값인 1을 가져옵니다.



2. Series

Series Slicing

- Series Slicing(시리즈 슬라이싱)은 Pandas 라이브러리에서 Series 객체의 일부를 선택하는 것을 의미합니다. 시리즈 슬라이싱은 인덱스를 기반으로 하여 특정 범위의 데이터를 추출하는 방법을 제공합니다.

SLICE	RESULT
0:1	First item
:1	First item (start default is 0)
:-2	처음부터 두 번째 항목부터 마지막 항목까지
::2	다른 모든 항목을 건너 뛰고 처음부터 끝까지 가져옵니다.



2. Series

Series Slicing

- 예를 들어, 다음과 같이 Series를 생성하고 슬라이싱하는 방법을 살펴보겠습니다.

파일	소스코드
실습환경	py3_10_basic
소스코드	<code>george_dupe.iloc[0:2]</code>
결과값1	1968 10 1969 7 Name: George Songs, dtype: int64
비고	



2. Series

Question 1

```
names = ['민준', '서연', '현우', '민서', '동현', '수빈']  
sdata = pd.Series(names)  
S1 = sdata[3:6]  
print(S1.values)  
print(S1)
```

의 결과는

```
['민서' '동현' '수빈']  
3    민서  
4    동현  
5    수빈  
dtype: object
```

이다. S1에서 '동현' 을 출력하는 코드로 맞는 것은?

- ① `print(S1[1])`
- ② `print(S1[4])`



2. Series

Answer 1

② `print(S1[4])`

#일반적인 인덱스 사용 시

```
b = S1[1:2]
```

```
print(b)
```

```
>>
```

```
4    동현
```


Boolean Arrays

- Boolean Arrays(불리언 배열)는 조건을 충족하는지 여부를 나타내기 위해 True 또는 False 값으로 이루어진 배열을 말합니다. Pandas에서는 불리언 배열을 사용하여 데이터를 필터링하거나 특정 조건에 따라 원하는 데이터를 선택하는 데 유용하게 활용됩니다.
- 예를 들어, 다음은 불리언 배열을 사용하여 데이터를 필터링하는 간단한 예제입니다.

파일	소스코드
실습환경	py3_10_basic
소스코드	<pre>mask = george_dupe > 7 mask</pre>
결과값1	<pre>1968 True 1969 False 1970 False 1970 True Name: George Songs, dtype: bool</pre>
비고	

Iteration

- 두 개의 Pandas Series를 생성하고, 첫 번째 Series인 songs_66의 각 값을 반복하며 출력하는 반복문입니다. 반복문에서 for value in songs_66:은 Series의 각 값을 차례로 반복하며, 해당 값들을 출력합니다. 그러나 이 코드는 None(결측치) 값을 가진 항목이 있기 때문에 에러가 발생할 수 있습니다.

파일	소스코드
실습환경	py3_10_basic
소스코드	<pre>songs_66 = pd.Series([3, None , 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts') for value in songs_66: print(value)</pre>
결과값1	<pre>3.0 nan 11.0 9.0</pre>
비고	



2. Series

Question 1

Pandas에서는 None(결측치) 대신에 NaN(Not a Number)을 사용하므로, None 값이 있는 Series를 반복할 때 에러를 피하려면 None을 NaN으로 변경하거나, None을 건너뛰도록 처리해야 합니다. songs_66 Series에서 None 값을 NaN으로 변경하여 반복하십시오.

Answer 1

```
import pandas as pd
import numpy as np

# 두 개의 Series 생성
songs_66 = pd.Series([3, None, 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts')
songs_66 = songs_66.replace({None: np.nan}) # None 값을 NaN으로 변경

# 반복문을 통해 songs_66의 각 값을 출력
for value in songs_66:
    print(value)

>>
3.0
nan
11.0
9.0
```



2. Series

Question 2

None 값이 있는 경우 해당 값이 None이면 출력하지 않도록 처리하시오.dropna() 메서드는 None이나 NaN 값을 제거하여 출력합니다.

Answer 2

```
import pandas as pd
```

```
# 두 개의 Series 생성
```

```
songs_66 = pd.Series([3, None, 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts')
```

```
# 반복문을 통해 songs_66의 각 값을 출력
```

```
for value in songs_66.dropna():  
    print(value)
```

```
>>
```

```
3.0
```

```
11.0
```

```
9.0
```

2. Series

Overloaded operations

- 두 개의 Pandas Series를 생성하고, 두 시리즈를 더한 후에 그 결과를 확인.
 - songs_66와 songs_69 두 시리즈의 각 요소를 동일한 인덱스끼리 더하여 새로운 Series를 만듭니다. 그러나 이 덧셈 연산에서 인덱스가 서로 다르게 배치되어 있기 때문에, 같은 인덱스를 가진 요소들끼리는 더해지고, 그렇지 않은 요소는 NaN(결측치)로 처리됩니다.

파일	소스코드
실습환경	<pre>py3_10_basic import pandas as pd</pre>
소스코드	<pre># 두 개의 Series 생성 songs_66 = pd.Series([3, None, 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts') songs_69 = pd.Series([18, 22, 7, 5], index=['John', 'Paul', 'George', 'Ringo'], name='Counts') # 두 Series를 더한 결과 result = songs_66 + songs_69 print(result)</pre>
결과값1	<pre>George 10.0 John 29.0 Paul 31.0 Ringo NaN Name: Counts, dtype: float64</pre>
비고	

Reset Index

- `reset_index()` 메서드는 Pandas Series나 DataFrame의 인덱스를 기본 숫자형 인덱스로 재설정하는 데 사용됩니다. 그러나 이 메서드는 호출된 Series나 DataFrame 자체를 변경하는 것이 아니라, 새로운 변경된 결과를 반환합니다.

파일	소스코드		
실습환경	py3_10_basic		
소스코드	<code>songs_66 = pd.Series([3, None, 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts')</code>		
	<code>songs_66.reset_index()</code>		
결과값1	0	index	Counts
	1	George	3.0
	2	Ringo	NaN
	3	John	11.0
비고	3	Paul	9.0

2. Series

Reset Index

- `songs_66.reset_index()`를 호출했을 때, `songs_66` Series의 인덱스는 변경되지 않고 그대로 유지됩니다. `reset_index()` 메서드를 사용하여 새로운 DataFrame 또는 Series를 생성하려면 반환된 결과를 새로운 변수에 할당해야 합니다. 예를 들어, `reset_index()`를 사용하여 새로운 DataFrame을 만들고 결과를 확인하는 방법은 다음과 같습니다.

```
import pandas as pd
```

```
songs_66 = pd.Series([3, None, 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts')
reset_songs_66 = songs_66.reset_index()
```

```
print(reset_songs_66)
```

```
>>
```

	index	Counts
0	George	3.0
1	Ringo	NaN
2	John	11.0
3	Paul	9.0

- 이렇게 하면 `reset_songs_66`은 기본 숫자형 인덱스로 재설정된 새로운 DataFrame 또는 Series가 됩니다. 결과를 출력하면 기존의 인덱스가 새로운 열로 추가된 것을 확인할 수 있습니다.

Series Methods

- Pandas Series에는 다양한 메서드가 내장되어 있어 데이터를 조작하고 분석하는 데 유용합니다. 이러한 메서드들은 데이터를 처리하고 조작하는 데 있어서 편리한 기능을 제공합니다.
- 아래는 Pandas Series에서 자주 사용되는 메서드 몇 가지입니다.
 - `head()` 및 `tail()`: Series의 처음 또는 끝 부분 일부를 보여줍니다.
`series.head()` # 처음 부분을 출력
`series.tail()` # 끝 부분을 출력
 - `describe()`: 요약 통계 정보를 제공합니다.
`series.describe()`
 - `value_counts()`: 각 값의 빈도수를 세어줍니다.
`series.value_counts()`
 - `unique()` 및 `nunique()`: 고유한 값의 배열을 반환하거나 고유한 값의 개수를 반환합니다.
 - `series.unique()` # 고유한 값들의 배열 반환
 - `series.nunique()` # 고유한 값의 개수 반환
 - `sort_values()`: 값에 따라 Series를 정렬합니다.
 - `series.sort_values()` # 값에 따라 정렬

Series Methods

- `fillna()`: 결측값을 다른 값으로 채웁니다.
`series.fillna(value)` # 결측값을 지정한 값으로 채움
- `isnull()` 및 `notnull()`: 각 요소가 결측값인지 아닌지를 확인합니다.
`series.isnull()` # 결측값이면 True, 아니면 False 반환
`series.notnull()` # 결측값이 아니면 True, 아니면 False 반환
- `apply()`: 함수를 Series의 각 요소에 적용합니다.
`series.apply(func)` # 지정한 함수를 각 요소에 적용
- `map()`: 딕셔너리와 같은 매핑을 사용하여 Series 값을 변환합니다.
`series.map(mapping_dict)` # 딕셔너리를 사용하여 Series 값을 변환
- `astype()`: Series의 데이터 타입을 변경합니다.
`series.astype(new_dtype)` # 데이터 타입을 지정한 타입으로 변경
- 이 외에도 Pandas Series는 다양한 유용한 메서드와 기능을 제공합니다. 이러한 메서드들은 데이터 조작과 분석에 유용하며, 데이터를 더 쉽게 처리할 수 있도록 도와줍니다.

2. Series

Series Methods

- Statistics

파일	소스코드
실습환경	<pre>py3_10_basic</pre>
	<pre>songs_66 = pd.Series([3, None , 11, 9], index=['George', 'Ringo', 'John', 'Paul'], name='Counts')</pre>
	<pre>songs_66</pre>
소스코드	<pre>songs_66.sum(0)</pre>
	<pre>songs_66.mean()</pre>
	<pre>songs_66.median()</pre>
	<pre>George 3.0 Ringo NaN John 11.0 Paul 9.0 Name: Counts, dtype: float64</pre>
결과값1	<pre>23.0 In []: 7.666666666666667 9.0</pre>
비고	

Series Methods

- Statistics

파일	소스코드
실습환경	py3_10_basic
소스코드	songs_66.quantile()
	songs_66.quantile(.1)
	songs_66.quantile(.9)
결과값1	songs_66.describe()
	9.0
	4.2
	10.6
	count 3.000000
	mean 7.666667
	std 4.163332
	min 3.000000
	25% 6.000000
	50% 9.000000
	75% 10.000000
	max 11.000000
	Name: Counts, dtype: float64

Series Methods

- Statistics

파일	소스코드
실습환경	py3_10_basic
	songs_66.min()
	songs_66.idxmin()
소스코드	songs_66.max()
	songs_66.idxmax()
	songs_66.var()
	songs_66.std()
결과값1	3.0
	'George'
	11.0
	'John'
	17.333333333333336
	4.163331998932266
비고	



3. DataFrame

개요

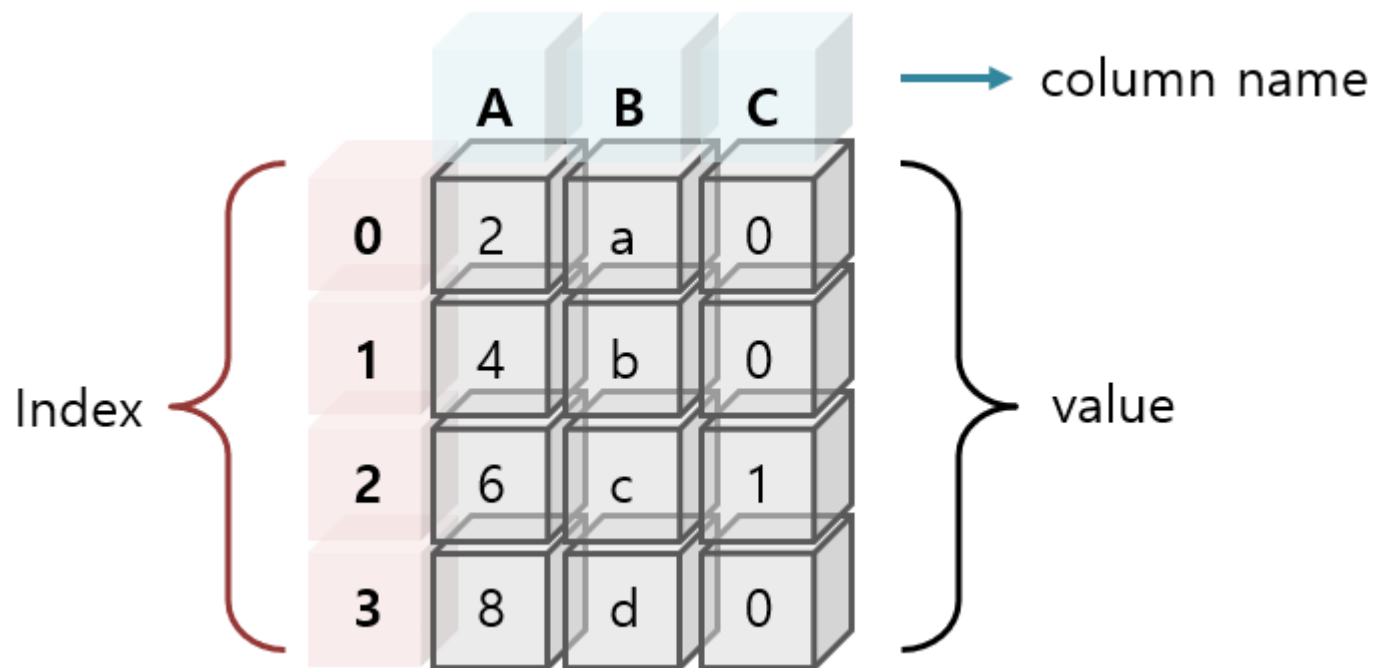
- Pandas 라이브러리의 기본 객체이자 2차원 객체인 데이터프레임(DataFrame) 객체는 2차원 배열과 유사하지만, 각각 열과 행에 라벨을 붙일 수 있다. 이 라벨을 통해 인덱싱(indexing)이 가능하며, 인덱싱을 통해 원소에 접근할 수 있다. 그리고 NumPy의 어레이 객체보다 강력하고 다양한 메서드를 지원하며 심지어 간단한 그래프도 쉽게 그릴 수 있다. 그리고 시리즈(Series) 객체를 여러 개 이어붙인 것이 데이터프레임이라고 할 수 있다.
- 데이터프레임은 엄밀히 말하면 인덱스가 행과 열에 각각 있는데 일반적으로 데이터프레임에서 인덱스라고 하면 행(row)의 인덱스를 뜻하며 열의 인덱스는 변수명(column name)으로 지칭한다. 그리고 인덱스/변수명/값 각 요소는 `.index/.columns/.values` 어트리뷰트로 접근할 수 있다.



3. DataFrame

개요

- Pandas 데이터프레임 구조





3. DataFrame

생성

- 데이터프레임 객체의 생성은 DataFrame() 함수에 리스트나 NumPy 어레이 객체를 입력으로 넣어 생성할 수 있으며 딕셔너리를 사용하여 생성하는 경우도 많다. 비어있는 데이터프레임 객체를 생성할 경우 함수 내부에 아무것도 쓰지 않는다. 데이터 프레임 생성 후 개별적으로 각 열 정의

파일	소스코드
실습환경	py3_10_basic
소스코드	import pandas as pd pd.DataFrame()
결과값1	
비고	

3. DataFrame

생성

- 데이터 프레임 생성 후 개별적으로 각 열 정의. 빈 데이터프레임을 만들고 'Name', 'Age', 'Driver' 열을 추가하여 데이터프레임을 확인합니다.

파일

소스코드

실습환경

py3_10_basic

```
# 라이브러리를 임포트합니다.  
import pandas as pd
```

```
# 데이터프레임을 만듭니다.  
dataframe = pd.DataFrame()
```

소스코드

```
# 열을 추가합니다.  
dataframe['Name'] = ['Jacky Jackson', 'Steven Stevenson']  
dataframe['Age'] = [38, 25]  
dataframe['Driver'] = [True, False]
```

```
# 데이터프레임을 확인합니다.  
dataframe
```

결과값1

	Name	Age	Driver
0	Jacky Jackson	38	True
1	Steven Stevenson	25	False

비고



3. DataFrame

생성

● 리스트 기반 생성

파일	소스코드				
실습환경	py3_10_basic				
소스코드	pd.DataFrame([[1, 2, 3]])				
결과값1	0	0	1	2	
			1	2	3
비고					

파일	소스코드		
실습환경	py3_10_basic		
소스코드	pd.DataFrame([[1, 2], [3, 4]])		
결과값1		0	1
	0	1	2
	1	3	4
비고			

3. DataFrame

생성

- 리스트 기반 생성
 - 원본 리스트 전달로 생성

파일	소스코드												
실습환경	py3_10_basic												
소스코드	<pre>import numpy as np import pandas as pd data = [['Jacky Jackson', 38, True], ['Steven Stevenson', 25, False]] pd.DataFrame(data, columns=['Name', 'Age', 'Driver'])</pre>												
결과값1	<table><tr><th></th><th>Name</th><th>Age</th><th>Driver</th></tr><tr><td>0</td><td>Jacky Jackson</td><td>38</td><td>True</td></tr><tr><td>1</td><td>Steven Stevenson</td><td>25</td><td>False</td></tr></table>		Name	Age	Driver	0	Jacky Jackson	38	True	1	Steven Stevenson	25	False
	Name	Age	Driver										
0	Jacky Jackson	38	True										
1	Steven Stevenson	25	False										
비고													



3. DataFrame

생성

- NumPy 어레이 기반 생성

파일	소스코드												
실습환경	py3_10_basic												
소스코드	<pre>import numpy as np import pandas as pd data = [['Jacky Jackson', 38, True], ['Steven Stevenson', 25, False]] matrix = np.array(data) pd.DataFrame(matrix, columns=['Name', 'Age', 'Driver'])</pre>												
결과값1	<table><tr><th></th><th>Name</th><th>Age</th><th>Driver</th></tr><tr><td>0</td><td>Jacky Jackson</td><td>38</td><td>True</td></tr><tr><td>1</td><td>Steven Stevenson</td><td>25</td><td>False</td></tr></table>		Name	Age	Driver	0	Jacky Jackson	38	True	1	Steven Stevenson	25	False
	Name	Age	Driver										
0	Jacky Jackson	38	True										
1	Steven Stevenson	25	False										
비고													

3. DataFrame

생성

- 딕셔너리 기반 생성
 - 딕셔너리의 키는 데이터프레임 객체의 변수명으로 된다. 열 이름과 매핑한 딕셔너리 사용 생성

파일	소스코드															
실습환경	py3_10_basic															
소스코드	<pre>data = {'Name': ['Jacky Jackson', 'Steven Stevenson'], 'Age': [38, 25], 'Driver': [True, False]} pd.DataFrame(data)</pre>															
결과값1	<table><tr><th>Name</th><th>Age</th><th>Driver</th><th></th><th></th></tr><tr><td>0</td><td>Jacky Jackson</td><td>38</td><td>True</td><td></td></tr><tr><td>1</td><td>Steven Stevenson</td><td></td><td>25</td><td>False</td></tr></table>	Name	Age	Driver			0	Jacky Jackson	38	True		1	Steven Stevenson		25	False
Name	Age	Driver														
0	Jacky Jackson	38	True													
1	Steven Stevenson		25	False												
비고																

- 열과 값을 매핑한 딕셔너리를 리스트로 전달해 생성

파일	소스코드												
실습환경	py3_10_basic												
소스코드	<pre>data = [{'Name': 'Jacky Jackson', 'Age': 38, 'Driver': True}, {'Name': 'Steven Stevenson', 'Age': 25, 'Driver': False}] pd.DataFrame(data, index=['row1', 'row2'])</pre>												
결과값1	<table><tr><th></th><th>Name</th><th>Age</th><th>Driver</th></tr><tr><td>row1</td><td>Jacky Jackson</td><td>38</td><td>True</td></tr><tr><td>row2</td><td>Steven Stevenson</td><td>25</td><td>False</td></tr></table>		Name	Age	Driver	row1	Jacky Jackson	38	True	row2	Steven Stevenson	25	False
	Name	Age	Driver										
row1	Jacky Jackson	38	True										
row2	Steven Stevenson	25	False										
비고													



2. Series

Question 1

새로운 Series를 DataFrame에 추가하는 다음 코드는 pandas 2.0.0 버전 이후부터 'append()' Method가 완전히 제거되었기 때문에 더 이상 작동하지 않습니다. 트러블 슈팅하시오.

```
import pandas as pd

# 데이터프레임을 만듭니다.
dataframe = pd.DataFrame()

# 열을 만듭니다.
new_person = pd.Series(['Molly Mooney', 40, True], index=['Name','Age','Driver'])

# 방법 1: append() 메서드 사용
dataframe = dataframe.append(new_person, ignore_index=True)

# 결과 확인
print(dataframe)
>>
```

AttributeError Traceback (most recent call last)

C...

AttributeError: 'DataFrame' object has no attribute 'append'



2. Series

Answer 1

```
import pandas as pd

# 데이터프레임을 만듭니다.
dataframe = pd.DataFrame()

# 열을 추가합니다.
dataframe['Name'] = ['Jacky Jackson', 'Steven Stevenson']
dataframe['Age'] = [38, 25]
dataframe['Driver'] = [True, False]

# 새로운 행을 만듭니다.
new_person = pd.Series(['Molly Mooney', 40, True], index=['Name', 'Age', 'Driver'])

# 방법 2: loc[]을 사용하여 새로운 행 추가
dataframe.loc[len(dataframe)] = new_person

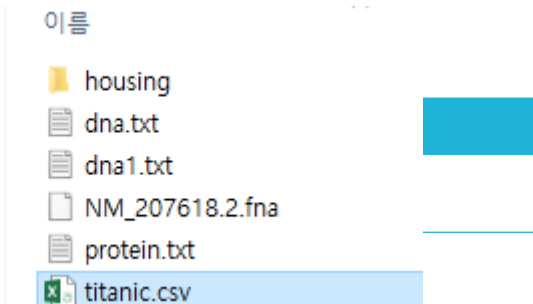
# 결과 확인
print(dataframe)
>>
```

	Name	Age	Driver
0	Jacky Jackson	38	True
1	Steven Stevenson	25	False
2	Molly Mooney	40	True

3. DataFrame

생성

● 데이터 가져오기 / 차원 확인

파일	소스코드	
실습환경	<pre>py3_10_basic</pre>	
	<pre># 라이브러리를 임포트합니다. import pandas as pd</pre>	
	<pre># 데이터를 적재합니다. dataframe = pd.read_csv("datasets//titanic.csv")</pre>	
소스코드	<pre># 두 개의 행을 확인합니다. dataframe.head(2)</pre>	
	<pre># 차원을 확인합니다. dataframe.shape</pre>	
결과값1	<pre>0 PassengerId Survived Pclass Parch Ticket Fare 1 1 0 3 A/5 21171 7.2500 Braund, Mr. Owen Harris male 22.0 2 2 1 1 female 38.0 1 Cumings, Mrs. John Bradley (Florence Briggs Th... PC 17599 71.2833 C85</pre>	
결과값2	<pre>(891, 12)</pre>	
비고		



3. DataFrame

DataFrame 생성과 몇 가지 검토

- 인덱스 지정

- 데이터프레임 객체 생성시 리스트나 어레이를 사용할 경우 변수명에 0부터 1씩 증가하는 값이 자동 부여되는데 직접 지정하고자 한다면 columns 인자에 값을 입력하면 된다. 텍스트가 직접적으로 표기된 것을 알 수 있다. 시리즈 객체와 마찬가지로 데이터프레임 또한 인덱스기반 연산도 가능하다.

파일	소스코드		
실습환경	py3_10_basic		
소스코드	<pre>pd.DataFrame([[1, 2], [3, 4]], columns = ["col1", "col2"])</pre>		
결과값1	0	col1	col2
	1	1	2
		3	4
비고			



3. DataFrame

DataFrame 생성과 몇 가지 검토

- 인덱스 지정
 - 인덱스를 지정하는 데이터프레임 객체 생성은 다음과 같다.

파일	소스코드			
실습환경	py3_10_basic			
소스코드	<pre>pd.DataFrame([[1, 2], [3, 4]], index = [100, 200])</pre>			
결과값1	100	1	0	1
	200	3	2	4
비고				

3. DataFrame

DataFrame 생성과 몇 가지 검토

- 주의점

- 데이터프레임을 생성할 때는 각 변수에 할당되는 원소의 개수가 같아야 된다. 만약 다음과 같은 코드로 변수를 생성하고자 한다면 에러가 발생하는 것을 볼 수 있다.

파일	소스코드
실습환경	py3_10_basic
소스코드	<pre>pd.DataFrame(dict(col1 = [1, 2], col2 = [3, 4, 5]))</pre>
결과값1	<pre>~\AppData\Roaming\Python\Python38\site-packages\pandas\core\internals\construction.py in _extract_index(data) 653 lengths = list(set(raw_lengths)) 654 if len(lengths) > 1: --> 655 raise ValueError("All arrays must be of the same length") 656 657 if have_dicts:</pre> <p>ValueError: All arrays must be of the same length</p>
비고	



3. DataFrame

Question 1

주어진 코드는 NumPy의 `randn()` 함수를 사용하여 3x4 형태의 랜덤한 숫자로 채워진 2차원 배열을 생성한 후, 이를 Pandas DataFrame으로 변환해 보자.

Answer 1

```
import numpy as np
import pandas as pd

# 3x4 형태의 랜덤한 숫자로 채워진 2차원 배열 생성
r = np.random.randn(3, 4)
print(r) # 생성된 랜덤 배열 출력

# 생성된 배열을 DataFrame으로 변환
d = pd.DataFrame(r, index=['one', 'two', 'three'], columns=['a', 'b', 'c', 'd'])
print(d) # DataFrame 출력
>>
[[-3.03230092  0.1323021  1.06509304 -1.11343787]
 [-0.13112835 -0.40795351  0.95316737  0.18322308]
 [ 0.06377442  0.54432592  0.94636602  0.29810242]]
      a         b         c         d
one  -3.032301  0.132302  1.065093 -1.113438
two   -0.131128 -0.407954  0.953167  0.183223
three  0.063774  0.544326  0.946366  0.298102
```



3. DataFrame

DataFrame 특정 행 선택과 조작

- DataFrame은 'state', 'year', 'pop' 열을 포함하고 있으며, 각 열에는 주어진 딕셔너리에 기반한 값들이 들어 있습니다. 여기서 'state'는 주의명, 'year'는 해당 연도, 'p

파일	소스코드
실습환경	py3_10_basic
소스코드	<pre>dic = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'], 'year': [2000, 2001, 2002, 2001, 2002], 'pop': [1.5, 1.7, 3.6, 2.4, 2.9]} d2 = pd.DataFrame(dic) d2</pre>

결과값1				
		state	year	pop
	0	Ohio	2000	1.5
	1	Ohio	2001	1.7
	2	Ohio	2002	3.6
	3	Nevada	2001	2.4
	4	Nevada	2002	2.9

비고



3. DataFrame

DataFrame 특정 행 선택과 조작

- DataFrame의 특정 속성을 사용하여 인덱스 및 열을 확인

파일	소스코드
실습환경	<pre>py3_10_basic pd.DataFrame(d2, columns = ['year', 'state', 'pop']) d2.index</pre>
소스코드	<pre>d2.columns 3 in d2.index 'state' in d2.columns</pre>
결과값1	<pre> year state pop 0 2000 Ohio 1.5 1 2001 Ohio 1.7 2 2002 Ohio 3.6 3 2001 Nevada 2.4 4 2002 Nevada 2.9</pre>
결과값1	<pre>RangeIndex(start=0, stop=5, step=1) Index(['state', 'year', 'pop'], dtype='object') True True</pre>
비고	



3. DataFrame

DataFrame 특정 행 선택과 조작

- DataFrame의 특정 속성을 사용하여 인덱스 및 열을 확인

파일

소스코드

1. `pd.DataFrame(d2, columns=['year', 'state', 'pop'])`는 d2 DataFrame에서 'year', 'state', 'pop' 열을 선택하여 새로운 DataFrame을 만듭니다.
2. `d2.index`는 DataFrame의 인덱스를 반환합니다.
3. `d2.columns`는 DataFrame의 열을 반환합니다.
4. `3 in d2.index`는 인덱스에 3이 있는지 여부를 확인합니다. (True 또는 False 반환)
5. `'state' in d2.columns`는 열에 'state'가 있는지 여부를 확인합니다. (True 또는 False 반환)

비고



3. DataFrame

DataFrame 특정 행 선택과 조작

- 열 선택
 - 열의 이름으로 선택

파일	소스코드
실습환경	py3_10_basic
소스코드	<pre>d['a'] d2['state']</pre>
결과값1	<pre>one -1.075684 two -0.159286 three -0.047937 Name: a, dtype: float64</pre>
결과값2	<pre>0 Ohio 1 Ohio 2 Ohio 3 Nevada 4 Nevada Name: state, dtype: object</pre>
비고	



3. DataFrame

DataFrame 특정 행 선택과 조작

- 열 삽입, 행 선택
 - `d2['debt'] = np.arange(5)` 코드는 DataFrame에 'debt'라는 이름의 열을 추가하고, 0부터 4까지의 값을 할당하는 것입니다.

파일	소스코드				
실습환경	py3_10_basic				
소스코드	d2['debt'] = np.arange(5)				
	d2				
	d2.loc[3]				
결과값1	d2.iloc[3]				
	0	state	year	pop	debt
	1	Ohio	2000	1.5	0
	2	Ohio	2001	1.7	1
	3	Ohio	2002	3.6	2
	4	Nevada	2001	2.4	3
결과값2	4				
	Nevada				
	2002				
	2.9				
	4				
결과값2	state Nevada				
	year 2001				
	pop 2.4				
	debt 3				
	Name: 3, dtype: object				
비고	state Nevada				
	year 2001				
	pop 2.4				
	debt 3				
	Name: 3, dtype: object				



3. DataFrame

DataFrame 특정 행 선택과 조작

● 행과 열 교체

- d2.T는 DataFrame의 전치(transpose)를 나타냅니다. DataFrame의 전치는 행과 열을 바꾸는 작업을 의미합니다.

파일	소스코드					
실습환경	py3_10_basic					
소스코드	d2					
	d2.T					
결과값1	0	state	year	pop	debt	
	1	Ohio	2000	1.5	0	
	2	Ohio	2001	1.7	1	
	3	Ohio	2002	3.6	2	
	4	Nevada	2001	2.4	3	
	5	Nevada	2002	2.9	4	
결과값2	state	0	1	2	3	4
	year	Ohio	Ohio	Ohio	Nevada	Nevada
	pop	2000	2001	2002	2001	2002
	debt	1.5	1.7	3.6	2.4	2.9
		0	1	2	3	4
비고						

3. DataFrame

DataFrame 특정 행 선택과 조작

- `reindex()` 메서드를 사용하여 새로운 인덱스로 DataFrame을 재색인하는 작업

파일	소스코드			
실습환경	<pre>py3_10_basic</pre>			
소스코드	<pre>data = np.arange(9).reshape((3, 3)) data</pre>			
	<pre>d3 = pd.DataFrame(data, index=['a', 'b', 'c'], columns=['Ohio', 'Texas', 'California']) d3</pre> <pre>d4 = d3.reindex(['a', 'b', 'c', 'd'], fill_value=0) d4</pre>			
결과값1	<pre>array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])</pre>			
결과값2		Ohio	Texas	California
	a	0	1	2
	b	3	4	5
결과값3	c	6	7	8
		Ohio	Texas	California
	a	0	1	2
	b	3	4	5
	c	6	7	8
	d	0	0	0
비고				



3. DataFrame

DataFrame 특정 행 선택과 조작

- `np.arange(9).reshape((3, 3))`는 0부터 8까지의 값을 포함하는 3x3 형태의 NumPy 배열을 생성합니다.
- `pd.DataFrame(data, index=['a', 'b', 'c'], columns=['Ohio', 'Texas', 'California'])`는 주어진 NumPy 배열을 사용하여 DataFrame을 생성합니다. 여기서 'a', 'b', 'c'는 행 인덱스이고, 'Ohio', 'Texas', 'California'는 열 이름입니다.
- `d3.reindex(['a', 'b', 'c', 'd'], fill_value=0)`는 d3 DataFrame을 'a', 'b', 'c', 'd'로 재색인(reindex)합니다. 'd'는 기존에 없던 인덱스이므로, `fill_value=0`을 사용하여 새로운 행을 추가하고 해당 행을 0으로 채웁니다.



3. DataFrame

DataFrame 특정 행 선택과 조작

- 행 또는 열 삭제

- 주어진 코드는 Pandas DataFrame에서 특정 행을 제거하는 방법을 보여줍니다. `drop()` 메서드는 DataFrame에서 특정 행이나 열을 제거하는데 사용됩니다.

파일	소스코드
----	------

실습환경	py3_10_basic
------	--------------

	<pre>py3_10_basic d5 = d4.drop('c') d5</pre>
--	--

소스코드	<pre>d5 = d4.drop(['c', 'd']) d5</pre>
------	--

결과값1	<pre>Ohio Texas California a 0 1 2 b 3 4 5 d 0 0 0</pre>
------	---

결과값2	<pre>Ohio Texas California a 0 1 2 b 3 4 5</pre>
------	--

비고	
----	--



3. DataFrame

DataFrame 특정 행 선택과 조작

- 행 또는 열 삭제

- 여기서 axis=1은 열을 나타내며, drop() 메서드에서 axis 매개변수를 사용하여 열을 삭제합니다. 기본적으로 axis=0으로 설정되어 있어 행을 삭제합니다. 만약 열을 삭제하려면 axis=1을 명시해야 합니다.

파일	소스코드		
실습환경	py3_10_basic		
소스코드	d5 = d4.drop('Ohio', axis=1)		
	d5		
	d5 = d4.drop(['Ohio', 'Texas'], axis=1)		
	d5		
결과값1		Texas	California
	a	1	2
	b	4	5
	c	7	8
	d	0	0
결과값2		California	
	a	2	
	b	5	
	c	8	
	d	0	
비고			



3. DataFrame

DataFrame 특정 행 선택과 조작

- Pandas DataFrame에서 `sort_index()` 메서드를 사용하여 인덱스 또는 열을 기준으로 정렬하는 방법을 보여줍니다.
 - `sort_index()` 메서드는 DataFrame의 행 또는 열을 인덱스 또는 열 이름을 기준으로 정렬하는 데 사용됩니다. `ascending=False`를 지정하면 내림차순으로 정렬됩니다. `axis=0`은 인덱스(행)을 기준으로 정렬하며, `axis=1`은 열을 기준으로 정렬합니다.

파일	소스코드				
실습환경	py3_10_basic				
	d4.sort_index(ascending=False) # DataFrame d4의 인덱스를 내림차순으로 정렬				
소스코드	d4.sort_index(axis=1) # DataFrame d4의 열 이름을 기준으로 정렬				
	d4.sort_index(ascending=False, axis=1) # DataFrame d4의 열 이름을 내림차순으로 정렬				
결과값1		Ohio	Texas	California	
	d	0	0	0	
	c	6	7	8	
	b	3	4	5	
	a	0	1	2	
결과값2			California	Ohio	Texas
	a	2	0	1	
	b	5	3	4	
	c	8	6	7	
	d	0	0	0	
결과값3		Texas	Ohio	California	
	a	1	0	2	
	b	4	3	5	
	c	7	6	8	
	d	0	0	0	
비고					

3. DataFrame

DataFrame Methods

- 합, 평균, 최소, 최대 구하기

파일	소스코드			
실습환경	py3_10_basic			
소스코드	<pre>d4 d4.sum() d4.sum(axis=1) d4.mean() d4.mean(axis=1)</pre>			
결과값1	a	Ohio	Texas	California
	b	0	1	2
	c	3	4	5
	d	6	7	8
		0	0	0
결과값2	<pre>Ohio 9 Texas 12 California 15 dtype: int64 a 3 b 12 c 21 d 0 dtype: int64</pre>			
결과값3	<pre>Ohio 2.25 Texas 3.00 California 3.75 dtype: float64 a 1.0 b 4.0 c 7.0 d 0.0</pre>			
비고				



4. Data Structures 외부 라이브러리

2. DataFrame

● 그래프

파일

소스코드

실습환경

py3_10_basic

소스코드

```
import pandas as pd
```

```
import matplotlib.pyplot as plt
```

```
import matplotlib
```

```
matplotlib.rcParams['font.family'] = 'Malgun Gothic' # Windows
```

```
# matplotlib.rcParams['font.family'] = 'AppleGothic' # Mac
```

```
matplotlib.rcParams['font.size'] = 15 # 글자 크기
```

```
matplotlib.rcParams['axes.unicode_minus'] = False # 한글 폰트 사용 시, 마이너스 글자가 깨지는 현상을 해결
```

```
score_pd=pd.read_csv('datasets/score.csv')
```

```
score_pd.head()
```

결과값1

	이름	학교	키	국어	영어	수학	과학
0	사회 홍길동	SW특기 강남고	197	90	85	100	95
1	85 박문수	Python 강남고	184	40	35	50	55
2	25 이순신	Java 강남고	168	80	75	70	80
3	75 임격정	Javascript 강남고	187	40	60	70	75
4	80 강백호	NaN 강남고	188	15	20	10	35
	10	NaN					

비고

4. Data Structures 외부 라이브러리

2. DataFrame

● 그래프

파일

소스코드

실습환경

data/score.csv

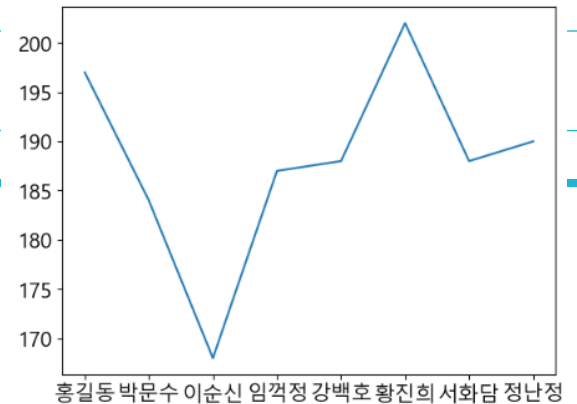
py3_10_basic

소스코드

```
plt.plot(score_pd['이름'], score_pd['키'])
```

결과값1

비고



4. Data Structures 외부 라이브러리

2. DataFrame

● 그래프

파일

소스코드

실습환경

data/score.csv

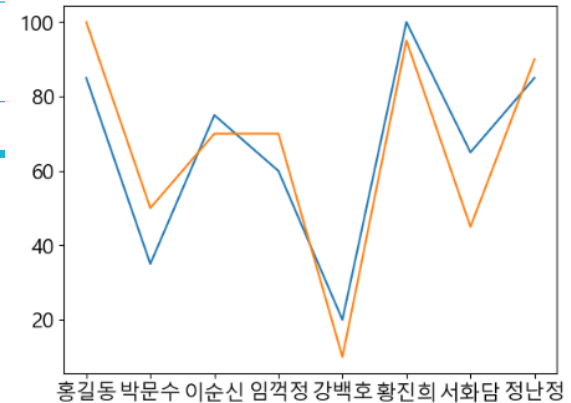
py3_10_basic

소스코드

```
plt.plot(score_pd['이름'], score_pd['영어'])  
plt.plot(score_pd['이름'], score_pd['수학'])
```

결과값1

비고



4. Data Structures 외부 라이브러리

2. DataFrame

● 그래프

파일

소스코드

실습환경

data/score.csv

py3_10_basic

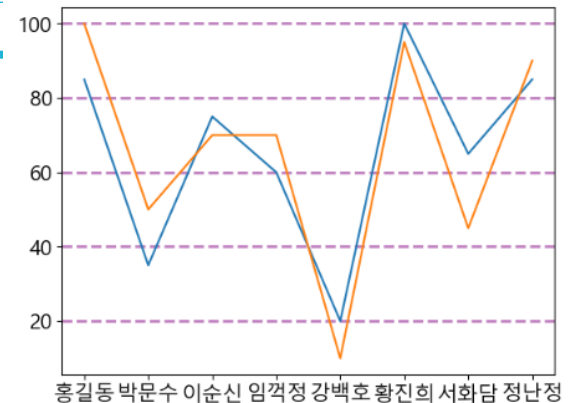
소스코드

```
plt.plot(score_pd['이름'], score_pd['영어'])  
plt.plot(score_pd['이름'], score_pd['수학'])
```

```
plt.grid(axis='y', color='purple', alpha=0.5, linestyle='--', linewidth=2)
```

결과값1

비고





4. Data Structures 외부 라이브러리

2. DataFrame

● 그래프

파일

소스코드

실습환경

data/score.csv

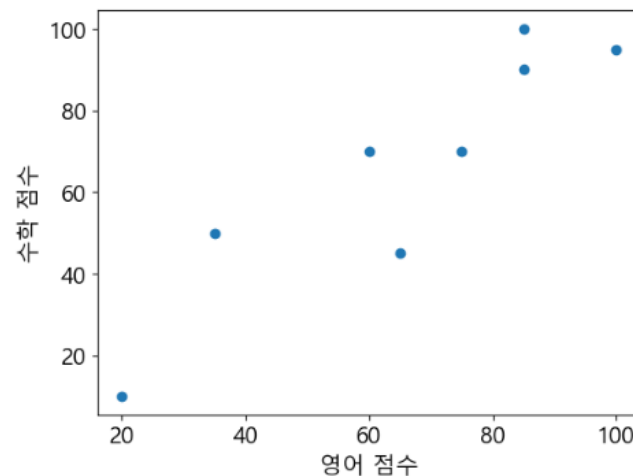
py3_10_basic

소스코드

```
plt.scatter(score_pd['영어'], score_pd['수학'])  
plt.xlabel('영어 점수')  
plt.ylabel('수학 점수')
```

결과값1

비고



4. Data Structures 외부 라이브러리

2. DataFrame

● 그래프

파일

소스코드

실습환경

data/score.csv

py3_10_basic

```
import numpy as np
sizes = np.random.rand(8) * 1000
sizes
```

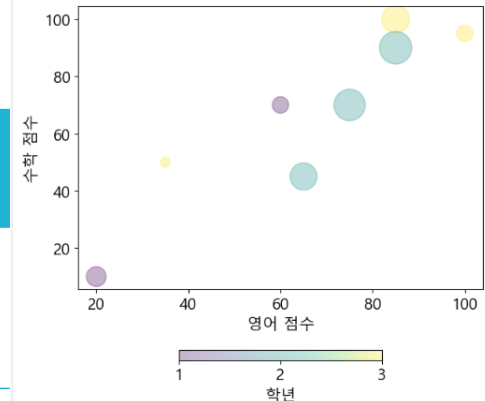
```
score_pd['학년'] = [3, 3, 2, 1, 1, 3, 2, 2]
```

소스코드

```
plt.figure(figsize=(7, 7))
plt.scatter(score_pd['영어'], score_pd['수학'], s=sizes, c=score_pd['학년'],
            cmap='viridis', alpha=0.3)
plt.xlabel('영어 점수')
plt.ylabel('수학 점수')
plt.colorbar(ticks=[1, 2, 3], label='학년', shrink=0.5, orientation='horizontal')
```

결과값1

비고





4. Data Structures 외부 라이브러리

2. DataFrame

● 그래프

파일

소스코드

실습환경

py3_10_basic

```
fig, axs = plt.subplots(2, 2, figsize=(15, 10)) # 2 x 2 에 해당하는 plot 들을 생성
fig.suptitle('여러 그래프 넣기')
```

```
# 첫 번째 그래프
```

```
axs[0, 0].bar(score_pd['이름'], score_pd['국어'], label='국어점수') # 데이터 설정
```

```
axs[0, 0].set_title('첫 번째 그래프') # 제목
```

```
axs[0, 0].legend() # 범례
```

```
axs[0, 0].set(xlabel='이름', ylabel='점수') # x, y 축 label
```

```
axs[0, 0].set_facecolor('lightyellow') # 전면 색
```

```
axs[0, 0].grid(linestyle='--', linewidth=0.5)
```

소스코드

```
# 두 번째 그래프
```

```
axs[0, 1].plot(score_pd['이름'], score_pd['수학'], label='수학')
```

```
axs[0, 1].plot(score_pd['이름'], score_pd['영어'], label='영어')
```

```
axs[0, 1].legend()
```

```
# 세 번째 그래프
```

```
axs[1, 0].barh(score_pd['이름'], score_pd['키'])
```

```
# 네 번째 그래프
```

```
axs[1, 1].plot(score_pd['이름'], score_pd['사회'], color='green', alpha=0.5)
```



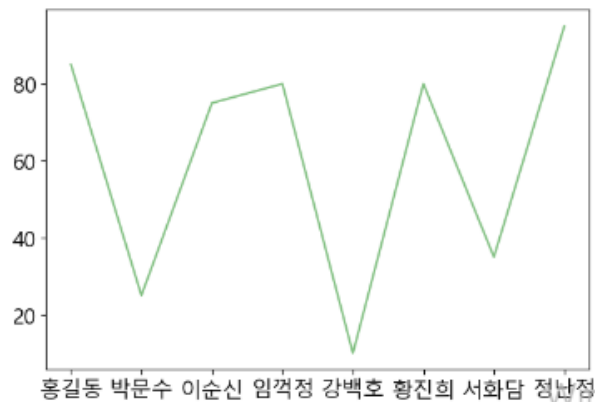
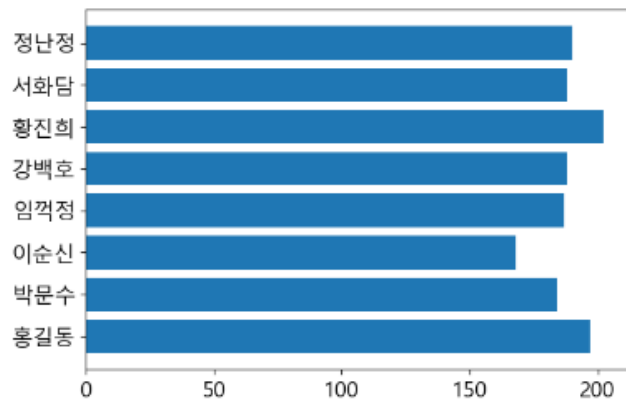
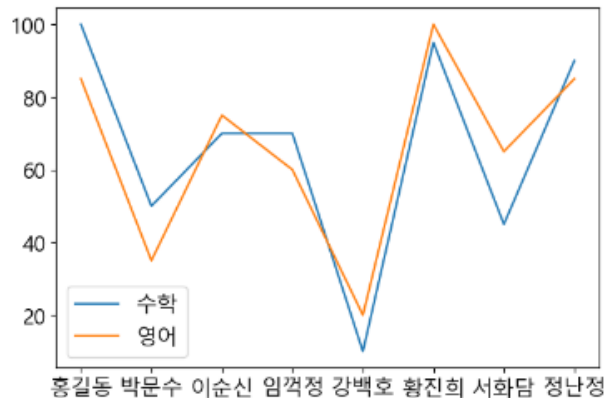
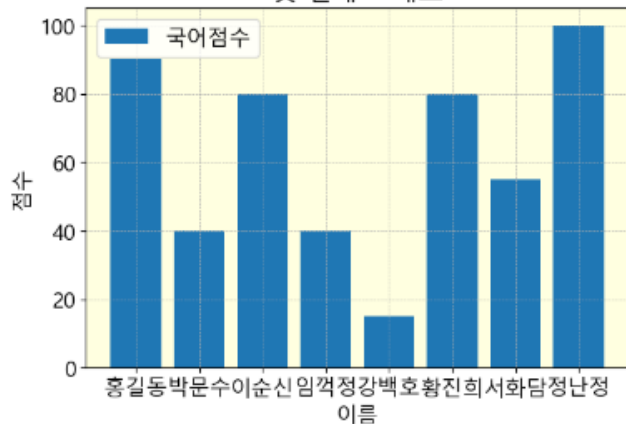
4. Data Structures 외부 라이브러리

2. DataFrame

● 그래프

여러 그래프 넣기

첫 번째 그래프



THANK YOU.

