데이터분석 및 시각화

데이터 분석(DATA ANALYSIS)

데이터 분석(DATA ANALYSIS)

◆ 데이터 분석이란?

유용한 **정보**를 **발굴**하고 **결론** 내용을 알리며
의사결정을 지원하는 것을 **목표**로
데이터를 정리, 변환, 모델링하는 과정이다.

과거의 데이터를 토대로 미래를 분석한다.

데이터 분석(DATA ANALYSIS)

◆ 데이터 분석 접근 방법

❖ 확증적 분석(CDA: Confirmatory Data Analysis)

- **전통적 분석 기법**으로 **추론 통계**에 주로 사용
- **가설 설정 후 수집 데이터로 가설 평가/추정**
- 기존 연구 기반으로 수행되되 엄격한 절차와 방법

- 장점 ➔ **검증된 이론/모형** 갖추고 있으며 **구체적인 질문/답 도출 가능**
- 단점 ➔ 선입견 개입되어 **예상치 못한 결과의 사전 탐지 어려움**

데이터 분석(DATA ANALYSIS)

◆ 데이터 분석 접근 방법

❖ 확증적 분석(CDA: Confirmatory Data Analysis)

가설 설정	데이터 수집	통계 분석	가설 검증
CCTV 설치하면 범죄 예방 효과 있다.	• 지역별 CCTV 설치 현황 데이터 • 지역별 범죄발생 기록 데이터	• CCTV 설치 지역과 범죄발생 빈도 상관관계 • CCTV 설치 전후 범죄발생 변화율	**분석결과** 통해 **가설을 채택하거나 기각**

데이터 분석(DATA ANALYSIS)

◆ 데이터 분석 접근 방법

❖ 탐색적 분석(EDA: Exploratory Data Analysis)

- 귀납적 분석 기법으로 기술 통계에 주로 사용
- 시각화 기법을 통해 데이터 특징, 구조로부터 통찰 얻는 기법

- 장점 ➔ 선입견 없이 **유연하게 데이터 분석하며 가설 설정 가능**
- 단점 ➔ 명확한 분석 **목표 없으면 방황할 가능성 높음**

데이터 분석(DATA ANALYSIS)

◆ 데이터 분석 접근 방법

❖ 탐색적 분석(EDA: Exploratory Data Analysis)

데이터 수집	시각화/탐색	패턴 도출	인사이트 발견
• 서울 지역별, 시기별 배달 음식 주문 건수 기록 데이터 확보	• 시각과 지역 변화에 따른 주문별 변화를 다양한 관점으로 시각화/탐색	• 시각화 자료로부터 음식별/시기별/지역별 일정한 패턴 있음 발견	• 시기와 지역별 주문이 많은 배달 음식 예상 • **창업에 활용**

데이터 분석(DATA ANALYSIS)

◆ EDA 5 단계

1 문제정의	2 데이터 수집	3 데이터 전처리
• 분석 대상 이해 • 객관적/구체적 대상 정의	• 필요 데이터 요건 정의 • 데이터 소재 파악/확보	• 오류 사항 점검 및 조치 • 데이터 구조 및 특성 변경

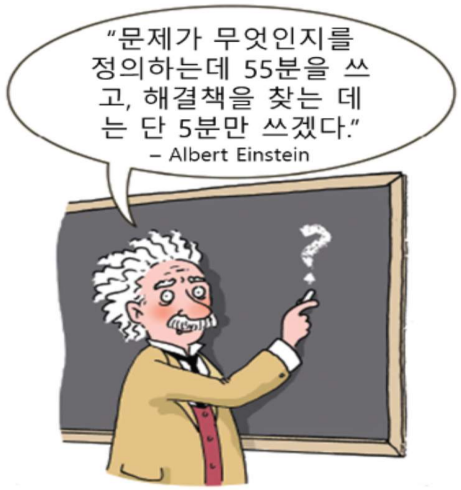
5 시각화 및 탐색	5 데이터 모델링
• 다양한 유형 시각화 • 문제 해결 인사이트 도출	• 다양한 관점 데이터 설계 • 관련 테이블간 관계 설정

데이터 분석(DATA ANALYSIS)

◆ EDA 5 단계

1 문제 정의

- **가장 중요 & 가장 어려운 단계**
  - 많은 사람들 공감할 만한 가치 있는 문제
  - 문제 해결위한 구체적인 행동 수반
  - 데이터 제약사항
  - 데이터 분석 전문가 및 기간 확보



아인시타인은 이렇게 말했다.

데이터 분석(DATA ANALYSIS)

◆ EDA 5 단계

2 데이터 수집

- **주변 >> 온라인 >> 오프라인**
  - 나의 PC / 나의 그룹 & 회사
  - 온라인 & 오프라인
  - 데이터 제공 기관 및 집단
    - ➢ 공공 : 공공데이터 포털, 통계빅데이터서비스, 한국복지패널
    - ➢ 지도 : 국가 공간정보 포털
    - ➢ 기상 : 기상자료개방포털
    - ➢ 관광 : 관광데이터랩
    - ➢ 건축 : 건축데이터 민간 개방 시스템, 국가공간정보포털, 등기정보광장

데이터 분석(DATA ANALYSIS)

◆ EDA 5 단계

3 데이터 전처리

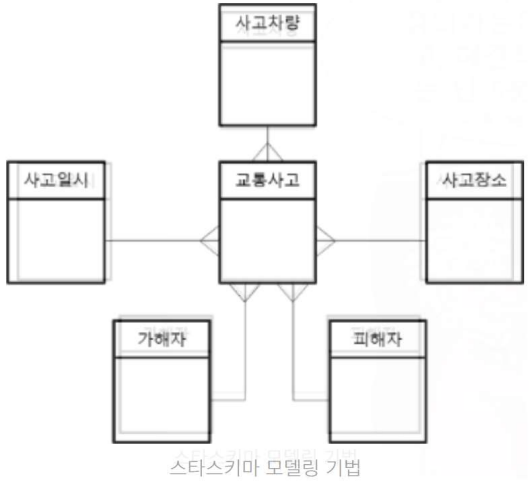
- **가장 많은 수고 및 시간 소요**
  - 수집 데이터를 그대로 사용하는 경우 거의 없음
    - ➢ 결측치 처리
    - ➢ 중복 처리
    - ➢ 이상치 처리
    - ➢ 데이터 연계 / 통합
    - ➢ 데이터 구조 변경

데이터 분석(DATA ANALYSIS)

◆ EDA 5 단계

4 데이터 모델링

- **관점별 데이터 분류 및 관계 설정**
  - 한 개의 핵심 사실(Fact)와 여러 개의 추가적인 사실로 구성
  - **분석 대상 나누어 그 결과를 사실과 추가 사실로 구성**하는 것
  - **데이터 설계과정**으로 **도식화 표현**
  - 모델링 예시)



스타스키마 모델링 기법

데이터 분석(DATA ANALYSIS)

◆ EDA 5 단계

5 시각화 및 탐색

- **패턴 찾고 인사이트 도출**
  - 문제에 대한 **답을 찾는 단계**
  - 대량 데이터 요약, 사람이 판단하기 쉬운 형태의 **이미지로 표현**
  - 이미지로 데이터에 숨겨진 ***유의미한 인사이트(Insite) 발견***할 수 있도록 도움
  - 데이터 요약 설명 방법 ➔ **기술 통계(Descriptive Statistics)**
    - 수집 데이터 **요약, 묘사, 설명**
    - **대표값(중심경향 : 평균, 중앙값, 최빈값 등등)과 분포** 이용해서 설명

데이터 분석(DATA ANALYSIS)

◆ 데이터 분류

- 데이터 종류에 따른 분석 방법 설정
- 데이터 분석 시 **제일 먼저 진행되어야 하는 분류**

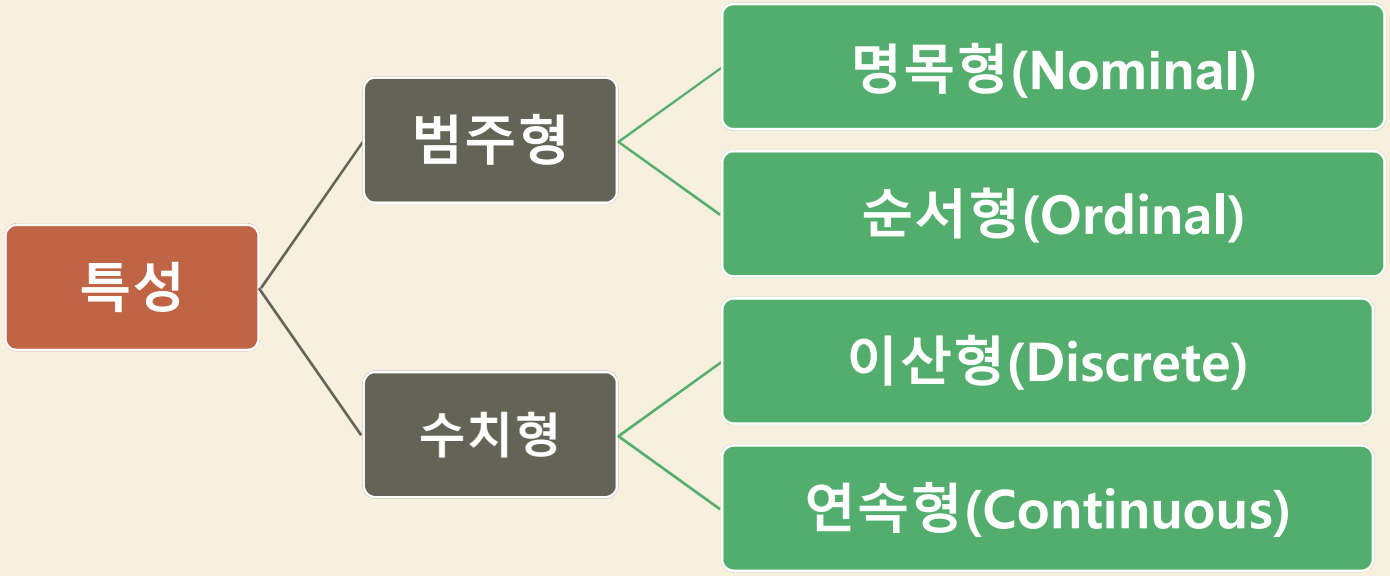
특성에 따른 분류

개수에 따른 분류

데이터 분석(DATA ANALYSIS)

◆ 데이터 분류

- **특성**에 따른 분류



데이터 분석(DATA ANALYSIS)

◆ 데이터 분류

▪ 특성에 따른 분류

범주형 데이터
(Categorical Data)

- **질적 (qualitative data , 정성적 자료)** 라고도 함
- **수치로 측정 불가능한 자료**
- 성별, 혈액형, 종교, 순위 등등 범주 또는 그룹으로 구분할 수 있는 데이터
  - ➢ **명목형** : 단순 분류 위해 사용되는 데이터. 성별, 혈액형 등 우선순위나 순서 없는 데이터
  - ➢ **순서형** : 서열이나 순위를 나타낼 수 있는 데이터, 학점,만족도

수치형 데이터
(Numerical Data)

- **양적 (quantitative data , 정량적 자료)** 라고도 함
- **수치로 측정 가능**한 자료
- 온도, 가격, 주가지수, 실업률, 매출액, 키 등등 숫자로 구성된 데이터
- 대소 비교, 평균, 최댓값, 최솟값 등 산술연산 분석 가능 데이터
  - ➢ **이산형** : 숫자가 정수로 딱딱 떨어지는 명확히 잘리는 형태
  - ➢ **연속형** : 값 사이에 무수히 많은 연속적인 값가진 형태

데이터 분석(DATA ANALYSIS)

◆ 데이터 분류

- **개수**에 따른 분류

변수 (Variable Data)	- 연구・조사・관찰하고 싶은 대상 - 예)키, 몸무게, 혈액형, 매출액, 습도, 먼지 농도 등등....
단일변수 자료 (Univariate Data)	- 연구・조사・관찰하고 싶은 대상이 **1개로만 구성된 자료** - **일변량 자료**라고도 함
다중변수 자료 (Multivariate Data)	- 연구・조사・관찰하고 싶은 대상이 **2개 이상으로 구성된 자료** - **다변량 자료**라도도 함

데이터 분석(DATA ANALYSIS)

◆ 데이터 분류

- 목적에 따른분류

목적/타겟/라벨/정답	- 데이터 분석에 목적이 되는 컬럼(열) - 예시) auto-mpg.csv 파일 ➔ mpg 컬럼
피쳐/특성/속성/데이터	- 목적 컬럼에 대한 부가 데이터이면서 증명해주는 데이터

데이터셋
(Dataset)

데이터 살펴보기

PANDAS 데이터 확인

◆ 데이터 정보 확인

>> 전체 데이터 구조 확인

>> 열 단위 데이터의 산술 통계 값 확인

>> 데이터의 형태, 크기(개수) 확인

>> 데이터 분석을 위한 기초 정확 수집

PANDAS 데이터 확인

◆ 데이터 정보 확인

>> 전체 데이터 구조 확인

>> 열 단위 데이터의 산술 통계 값 확인

>> 데이터의 형태, 크기(개수) 확인

>> 데이터 분석을 위한 기초 정확 수집

PANDAS 데이터 확인

◆ 데이터 정보 확인

➢ 정보 함수

앞부분 데이터 보기 : DataFrame 객체. **head (n) 기본 5개**

뒷부분 데이터 보기 : DataFrame 객체. **tail (n)**

데이터 기본 정보 보기 : DataFrame 객체. **info()**

데이터 기술 통계정보 : DataFrame 객체. **describe()**

데이터 기술 통계정보 : DataFrame 객체. **describe(include ='all')**

PANDAS 데이터 확인

◆ 데이터 정보 확인

➢ 데이터 개수 함수

전체 열 각 데이터 개수 반환 = DataFrame 객체**. count()**

고유값 종류&개수 반환 = DataFrame객체['열이름']. **value_counts()**

- 옵션 : dropna = False (기본값)

결측치(Missing value , NaN) 포함 여부

PANDAS 데이터 확인

◆ 데이터 통계 개념

➢ 값 사이 연관성 & 분포 여부

- **편차 (Deviation)**

  관측값에서 평균(mean)을 뺀 값, 평균에서 얼마나 떨어진 값인지 확인

- **분산(Variance)**

  평균(mean)에서 얼만큼 벗어나 있는가를 나타내는 값 , 편차를 제곱한 값

- **표준편차(standard deviation)**

  분산 값에 루트를 씌운 값

PANDAS 데이터 확인

◆ 데이터 통계 개념

➢ 값 사이 연관성 & 분포 여부

A 1,3,5,7,9

편차	-4	-2	0	2	4
편차의 제곱	16	4	0	4	16

분산 (편차의 제곱의 평균) $(16+4+0+4+16) \div 5 = 8$

표준 편차 (분산의 제곱근) $\sqrt{8}$

B 3,4,5,6,7

편차	-2	-1	0	1	2
편차의 제곱	4	1	0	1	4

분산 (편차의 제곱의 평균) $(4+1+0+1+4) \div 5 = 2$

표준 편차 (분산의 제곱근) $\sqrt{2}$

PANDAS 데이터 확인

◆ 데이터 통계 개념

➢ 대표값

- **평균 (mean)**

  모든 관측값의 합을 자료의 개수로 나눈 값

- **중앙(간)값(median)**

  전체 관측값을 크기 순서로 정렬했을 때 가운데 위치한 값

  홀수 경우: (n+1) /2 번째

  짝수 경우: (n/2)번째 관측값과 (n+1) /2번째 관측값의 평균

- **최빈값 (mode)**

  데이터 중 가장 많은 빈도로 나타나는 값

PANDAS 데이터 확인

◆ 데이터 통계 개념

➢ 분포(Distribution)

- **데이터**는 언뜻 보면 아무 의미 없는것 같지만, 만약 **어떤 분포를 따르는 잘 알려진 현상(상황)이라면 이는 분포를 통해 다른 현상들을 해석하거나 예측할 수 있게 되는 것**

- 대표적인 분포의 종류
  - 이산확률분포 ➔ 이항분포, 베르누아 분포, 기하분포, 푸아송 분포
  - 연속확률분포 ➔ 정규(Z)분포, T분포, F 분포, χ2분포

PANDAS 데이터 확인

◆ 데이터 통계 개념

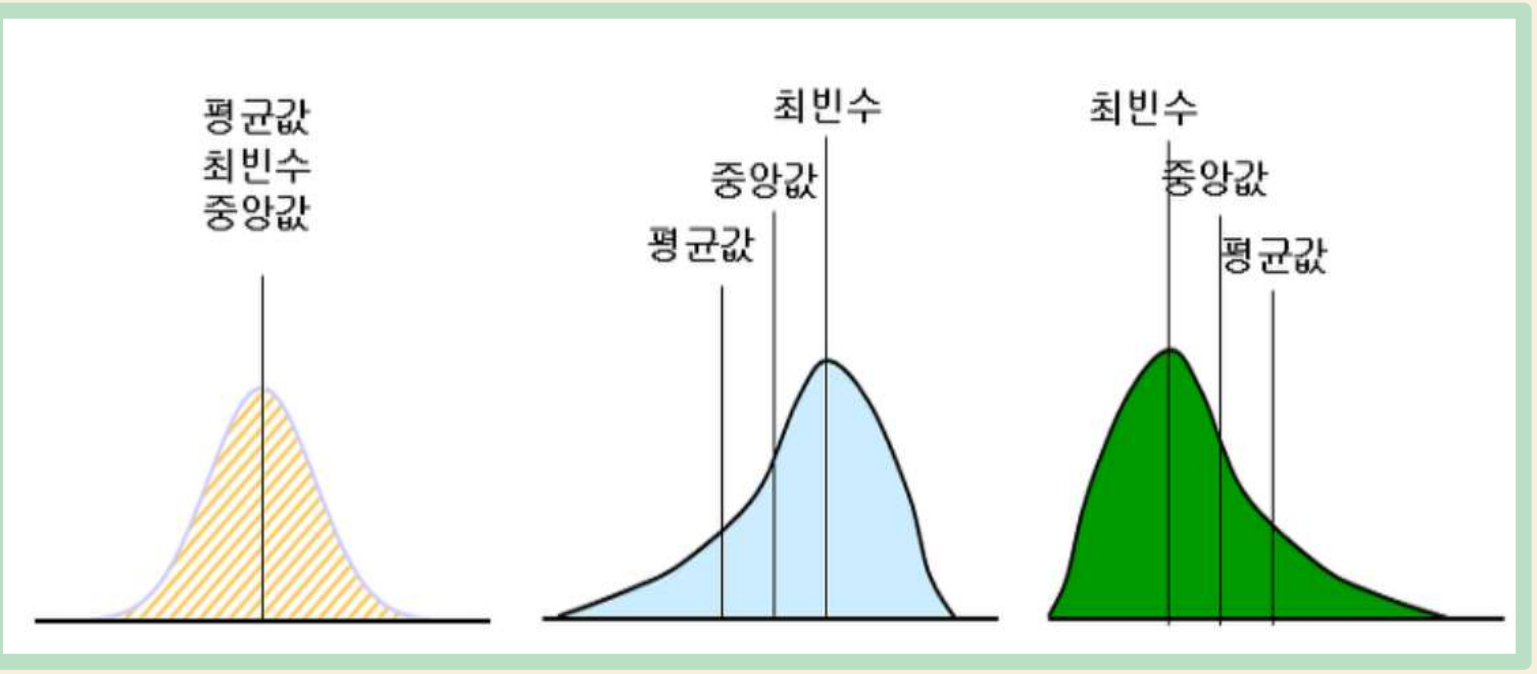
➢ 정규분포(Normal Distribution) ← 수학

- **가우시안(Gaussian) 분포**라고도 함 ← 공학
- **수집된 자료의 분포를 근사하는 데에 자주 사용**
- 중심극한정리에 의하여 독립적인 확률변수들의 평균은 정규분포에 가까워지는 성질이 있기 때문
- 평균, 최빈값 일치/평균 중심으로 좌우 대칭 → 평균, 중앙값 일치
- **평균 == 최빈값 == 중앙값**

PANDAS 데이터 확인

◆ 데이터 통계 개념

➢ 분포(Distribution)



PANDAS 데이터 확인

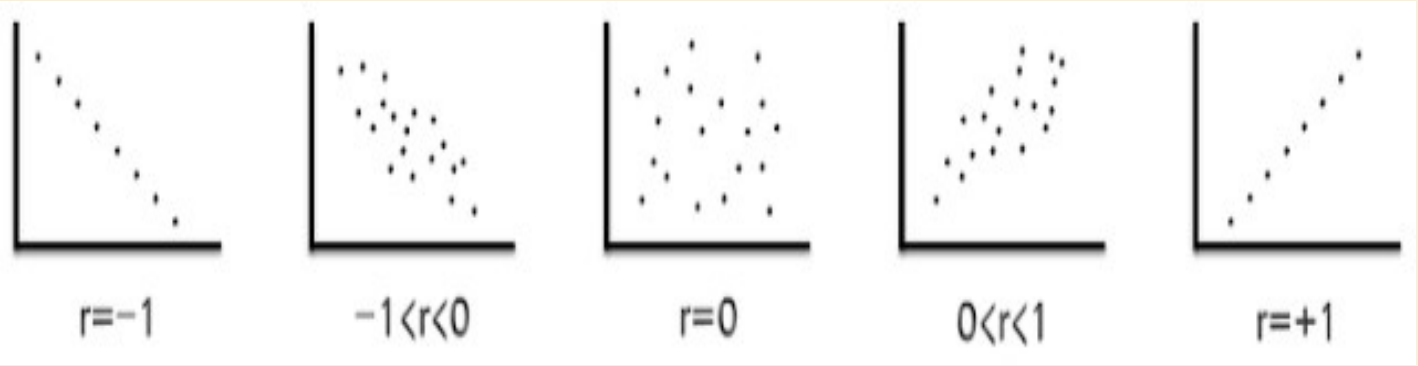
◆ 데이터 통계 개념

➢ 값들의 관계

- **상관계수(correlation coefficient)**

  두 열(column) 사이 관계 정도를 나타내는 수치

  - 범위 : -1 ~ 1 사이
  - -1(음의 상관관계), 1(양의 상관관계)에 가까울 수록 관계 밀접



PANDAS 데이터 확인

◆ 데이터 정보 확인

➢ 통계 함수

모든 열 평균값 = DataFrame 객체.**mean()**
특정 열 평균값 = DataFrame 객체**['열이름']. mean()**

모든 열 중간값 = DataFrame 객체.**median()**
특정 열 중간값 = DataFrame 객체**['열이름']. median()**

모든 열 최대값 = DataFrame 객체.**max()**
특정 열 최대값 = DataFrame 객체**['열이름']. max()**

모든 열 최소값 = DataFrame 객체.**min()**
특정 열 최소값 = DataFrame 객체**['열이름']. min()**

PANDAS 데이터 확인

◆ 데이터 정보 확인

➢ 통계 함수

모든 열 표준편차 = DataFrame 객체. **std()**

특정 열 표준편차 = DataFrame 객체**['열이름']. std()**

모든 열 상관계수 = DataFrame 객체.**corr()**

특정 열 상관계수 = DataFrame 객체**['열이름']. corr()**

PANDAS 데이터 확인

◆ 데이터 정보 확인

➢ 통계 개념

- **결측값(Missing Value)**

  입력되지 않은 값

  - 표기 : NaN
  - 사용 : 분석에서 제거 & 다른 값 치환

- **특이값(Outlier)**

  정상적인 데이터 분포 범위 박에 존재하는 값

  - 논리적으로 존재할 수 없는 값 (예: 나이 -7, 1002)

데이터 사전처리

PANDAS 데이터 사전처리

◆ 사전처리(Preprocessing)

>> **데이터 품질**을 높이기 위한 과정

>> **분석 목적에 맞게 변형**하는 과정

>> **데이터 처리** => 누락 데이터, 중복 데이터

>> **데이터 변환** => 단위 표준화, 자료형 표준화

>> **데이터 크기** => 정규화

PANDAS 데이터 사전처리

◆ 누락 데이터 처리

- 누락 데이터(Missing Value)란?

  - 데이터 입력 시 **미입력된 데이터**
  - 파일 형식 변경 과정에서 **데이터 소실**

  - 표기
    - numpy.NaN (Not a Number) : 표현되지 않는 부동소수점 값
    - pandas.NA (Not Avaiable) : 일반적인 결측값
    - pandas.NaT (Not a Time) : 날짜시간에 대한 결측값
    - None : Python에서 존재하지 않음 의미 즉, 비어있음

PANDAS 데이터 사전처리

◆ 누락 데이터 처리

- 누락 데이터 여부(True/False) 체크 함수

bool = 객체.**isnull()**
bool = 객체.**isna()**

➔ NA면 True, 아니면 False

bool = 객체.**notnull()**
bool = 객체.**notna()**

➔ NA 아니면 True, 맞으면 False

PANDAS 데이터 사전처리

◆ 누락 데이터 처리

- 누락 데이터 여부(True, False) 체크 함수

객체.isnull()

	survived	pclass	sex	age	...	deck	embark_town	alive	alone
0	False	False	False	False	...	True	False	False	False
1	False	False	False	False	...	False	False	False	False
2	False	False	False	False	...	True	False	False	False
3	False	False	False	False	...	False	False	False	False
4	False	False	False	False	...	True	False	False	False

PANDAS 데이터 사전처리

◆ 누락 데이터 처리

- 누락 데이터 개수

객체.isnull().sum()

	survived	pclass	sex	age	...	deck	embark_town	alive	alone
0	0	0	0	177	...	688	2	0	0

PANDAS 데이터 사전처리

◆ 누락 데이터 처리

- 누락 데이터 **제거 함수**

객체.dropna( ) ➔ 결측값 존재 행/열 삭제

• axis = 0	행(row) 삭제
• axis = 1	열(column) 삭제
• how = 'all'	모든 값 결측치 인 경우 삭제
• how = 'any'	하나라도 결측치 인 경우 삭제 (기본)
• thresh = 숫자	Na개 이상, 임계치 설정
• subset = 조건	하위 조건

PANDAS 데이터 사전처리

◆ 누락 데이터 처리

- 누락 데이터 **치환 함수**

객체.fillna(치환값) ➔ 특정값, 평균값, 최빈값 등으로 치환

value = 특정값

- fillna(0) : 모두 동일 값
- fillna({ 컬럼: 0, 컬럼: 1, 컬럼: 2, 컬럼: 3} : 컬럼별 다른 값
- fillna(평균값)
- fillna(최빈값)
- fillna(중앙값)

PANDAS 데이터 사전처리

◆ 누락 데이터 처리

- 누락 데이터 **치환 함수**

객체.fillna(치환값)

method = 'ffill' / 'pad'	직전 행(row) 값으로 치환
'bfill' / 'backfill'	바로 다음 행(row) 값으로 치환

limit = int	지정된 수 만큼만 치환
None	모두 치환

PANDAS 데이터 사전처리

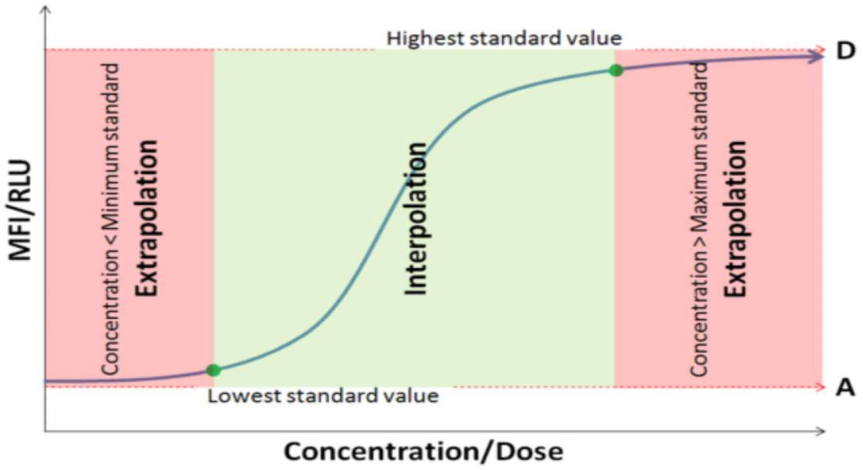
◆ 누락 데이터 처리

- 누락 데이터 **치환 함수**

객체. interpolate(치환값)

- 보간법/내삽

범위 안에 있는 값



PANDAS 데이터 사전처리

◆ 중복 데이터 처리

하나의 행(row) ➔ 모든 속성(coluum) 값 존재

완전한 데이터 (레코드 또는 관측값)

PANDAS 데이터 사전처리

◆ 중복 데이터 처리

- 중복 데이터 여부 함수

  객체.**duplicated()** ➔ 중복이면 True, 아니면 False

- 중복 제거 함수

  객체.**drop_duplicates()** ➔ 중복 제거

PANDAS 데이터 사전처리

◆ 이상치 데이터 처리

▪ 이상치 데이터(Outlier Value) ?

- 관측 **데이터 범위에서 많이 벗어난 아주 작은 값/ 큰 값**
- **일반적인 데이터 분포를 따르지 않는 값**
- 발생원인
  - 데이터 수집 과정에서 오류 발생한 경우
  - 데이터 자체에 이상치 포함한 경우 (오염된 데이터)
  - 값을 잘못 옮겨적을 경우
  - 실험 과정의 오류
  - 의도적인 자료의 조작

PANDAS 데이터 사전처리

◆ 이상치 데이터 처리

▪ 처리 방법

- 삭제 (Delete) / 절단
- 대체 (Replacement) / 조정
- 축소/과장(Scaling) 적용
- 최소초대척도 적용
- 정규화 적용

PANDAS 데이터 사전처리

◆ 이상치 데이터 처리

▪ 처리 방법 1) 삭제(Deleting) / 절단(Trimming)

- 극단적으로 크거나 작은 값 제거
- 극단적 값의 유의미한 경우 많음
- **단점 및 주의!! 삭제로 유의미한 데이터 손실 발생**

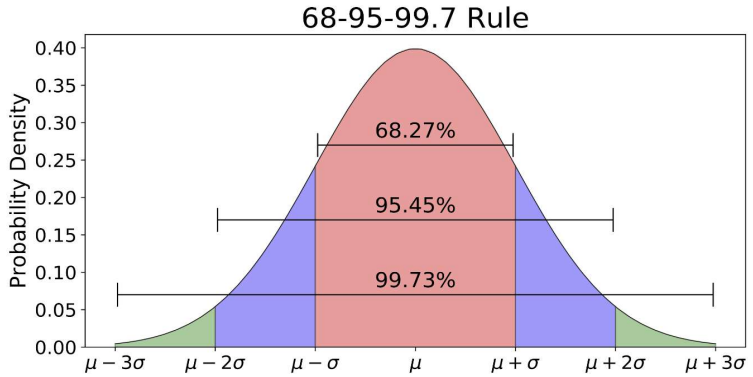
  ➔ 나머지 방법들 많이 사용

PANDAS 데이터 사전처리

◆ 이상치 데이터 처리

▪ 처리 방법 2) 조정(Winsorizing)

- ESD(Estreme Studentized Deviate test)
  - **평균으로부터 3표준편차 떨어진 값** 이상치 인식
  - 하한값/상한값으로 대체
  - 적합한 방법 아님



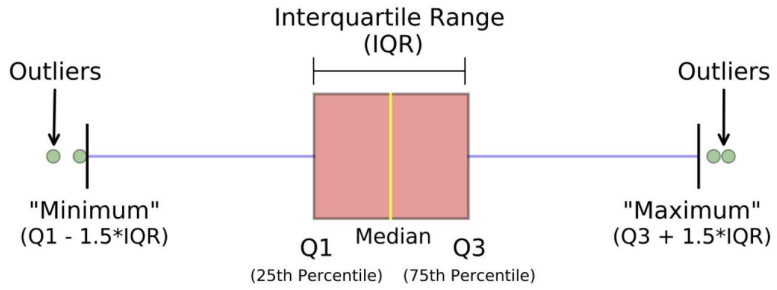
* 출처 : https://towardsdatascience.com

PANDAS 데이터 사전처리

◆ 이상치 데이터 처리

▪ 처리 방법 2) 조정(Winsorizing)

- 사분위수 : Quartiles
  - 데이터를 동일한 4개 부분으로 나눈 후 **상/하한값**
  - 적합한 방법 아님

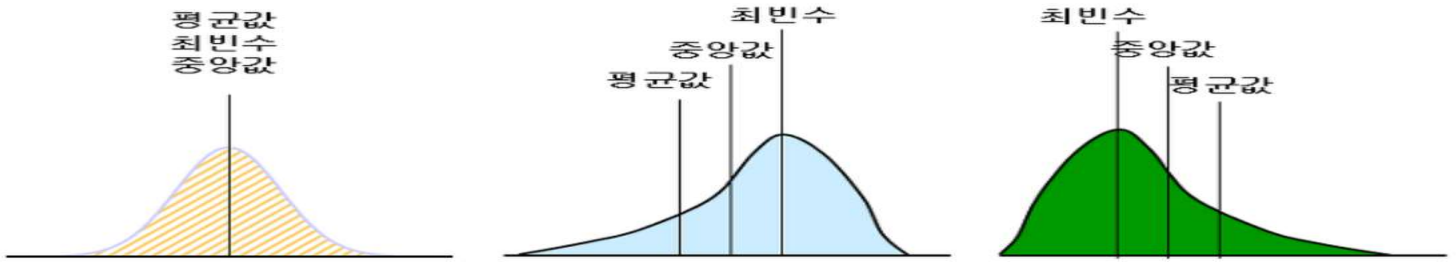


PANDAS 데이터 사전처리

◆ 이상치 데이터 처리

▪ 처리 방법 3) 축소/과장

- **데이터의 쏠림 정도 확인** 후 **데이터 분포를 고르게 만듦**
  - DataFrame['컬럼명'].skew()
    - 양수 : 평균보다 작은 값 데이터 많음 최빈<중앙<평균
    - 0 : 평균 중심 고르게 분포 최빈==중앙==평균
    - 음수 : 평균보가 큰 값 데이터 많음 평균<중앙<최빈



PANDAS 데이터 사전처리

◆ 이상치 데이터 처리

▪ 처리 방법 3) 축소/과장

- **Positive/Right Skew >> 값 축소 >> skew() 값 0 근접**
  - 로그값 / 제곱근값 처리

- **Negative/Left Skew >> 값 과장 >> skew() 값 0 근접**
  - 제곱 / 지수곱 처리

PANDAS 데이터 사전처리

◆ 이상치 데이터 처리

▪ 처리 방법 4) 최소초대척도(MinMax Scaling)

- 최대값 : 1, 최소값 : 0 변환
- 각 구간값을 0 ~ 1 사이로 스케일링

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

PANDAS 데이터 사전처리

◆ 이상치 데이터 처리

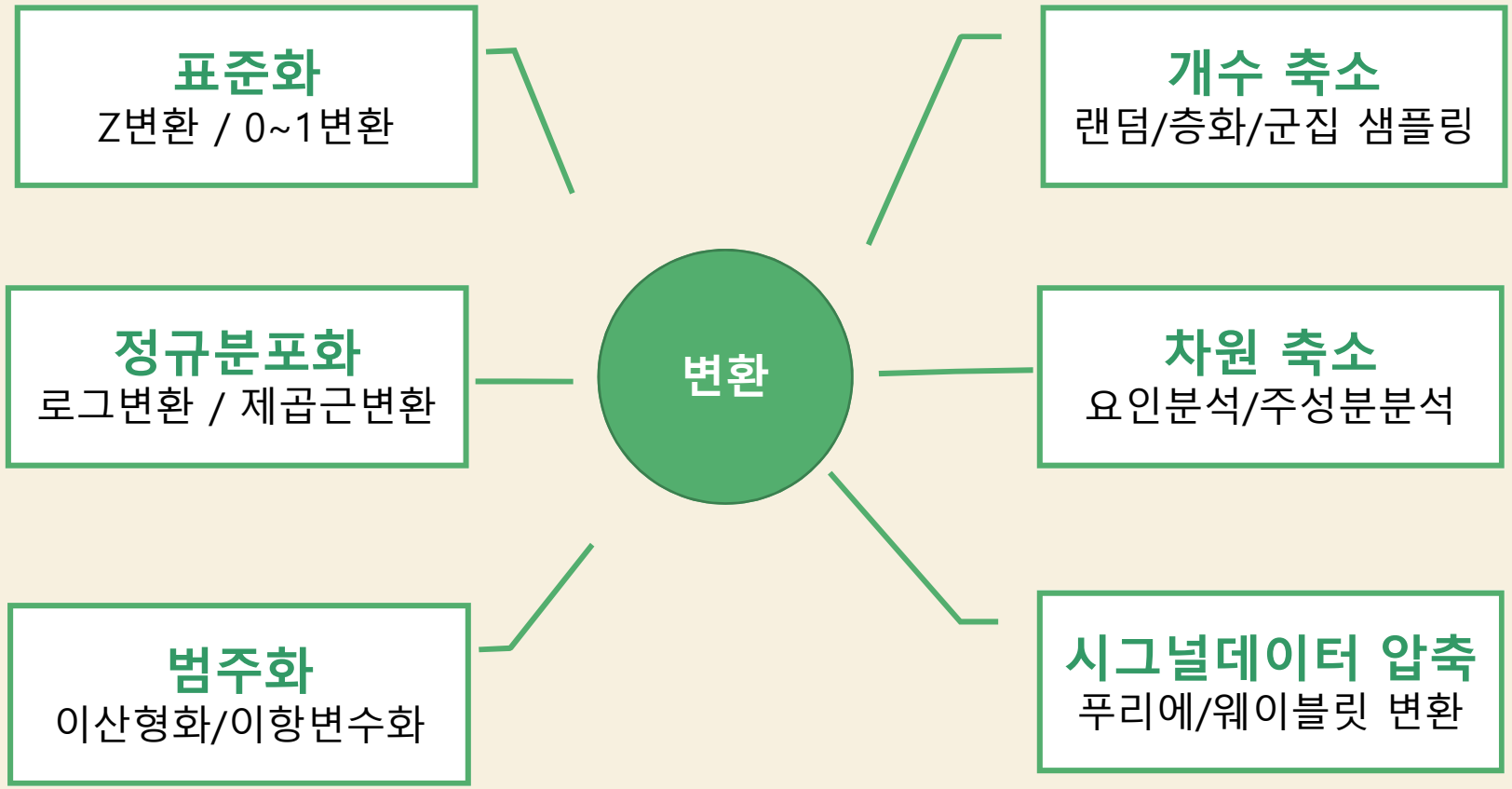
▪ 처리 방법 5) 정규화

- Z-Score
  - 평균 0, 표준편차 1

$$z = \frac{\mathrm{X} - mean(x)}{stdev(x)}$$

PANDAS 데이터 사전처리

◆ 데이터 변환(Transformation)



PANDAS 데이터 사전처리

◆ 데이터 표준화

➢ 표준화(Standardization)란?

- 수집 & 정리된 **데이터의 동일 포맷** 작업
- 여러 가지 제품들 종류/규격을 **표준 따라 제한/통일**하는 것
- 데이터 **분석 시 정확도 높임**
- 출처 : 위키백과 https://ko.wikipedia.org/wiki

➢ 방법

- 단위 환산 / 자료형 변환 / 범위 변환

PANDAS 데이터 사전처리

◆ 데이터 표준화

▪ 방법1) 단위 환산

- 수집 데이터의 **나라별 사용 단위 상이**
- **단위별 변환 방식**으로 환산
- 환산된 새로운 **열(column) 추가**

PANDAS 데이터 사전처리

◆ 데이터 표준화

➢ 방법 2) 자료형 변환

- 기본 자료형 => int, float, str, object
- 데이터 자료형 => 범주형, 연속형

▪ 함수

객체.astype('자료형') ➔ 자료형으로 변환

객체.dtypes ➔ 자료형 확인

PANDAS 데이터 사전처리

◆ 데이터 표준화

➢ 방법 2) 자료형 변환 : 연속형>>>범주형 : 이산화

연속형 데이터 일정 구간 ➔ 범주형 데이터 변환

```
pandas.cut((x=df[컬럼명],                  # 데이터 배열
            bins=n,                        # 경계 값 리스트
            labels=bin_names,              # bin 이름
            include_lowest=True)           # 첫 경계값 포함
)
```

PANDAS 데이터 사전처리

◆ 데이터 표준화

- 방법 2) 자료형 변환 : 범주형 >>> 수치형 : **인코딩**

❖ **라벨 인코딩**

- 알파벳 순서에 따라 문자형 데이터를 unique한 숫자형으로 매핑

ID	과일
1	사과
2	바나나
3	체리

LabelEncoder

ID	과일
1	0
2	1
3	2

PANDAS 데이터 사전처리

◆ 데이터 표준화

- 방법 2) 자료형 변환 : 범주형 >>> 수치형 : **인코딩**

❖ **원 핫 인코딩 (One Hot Encoding)**

- 라벨 수 만큼 자릿수 만든 해 해당 자리에 있으면 0, 없으면 1

pandas.get_dummies(구간분할 데이터)

ID	과일
1	사과
2	바나나
3	체리

One-Hot Encoding

ID	사과	바나나	체리
1	1	0	0
2	0	1	0
3	0	0	1

PANDAS 데이터 사전처리

◆ 데이터 표준화

- 방법 3) 범위 변환 : 연속형 데이터 값 범위 Scaling

  - **각 열(Feature)**의 **숫자 데이터 상대적 크기 차이 해결**
  - **동일한** 크기 기준**으로 나눈 비율**로 나타내는 것
  - 데이터 범위 : **0 ~ 1** 또는 **-1 ~ 1**

PANDAS 데이터 사전처리

◆ 데이터 정규화

- 방법 3) 범위 변환 >>> 정규화(Normalization)

  - **입력된 x 값들을 모두 0과 1사이의 값으로 변환하는 방법**
  - Min-Max Normalization(최소-최대 정규화)
  - 범위 : 0 ~ 1

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

PANDAS 데이터 사전처리

◆ 데이터 정규화

▪ 방법 3) 범위 변환 >>> 표준화(Standardization)

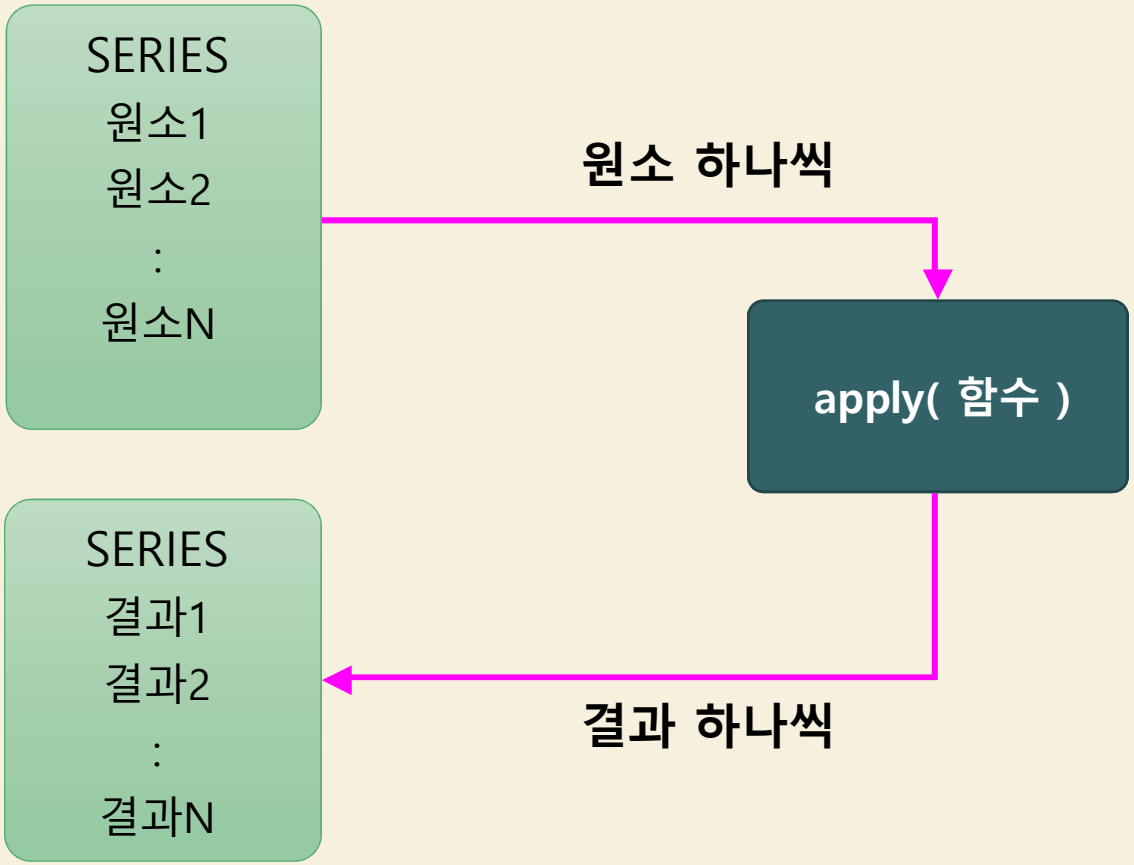
- **데이터들이 정규 분포를 갖도록 평균이 0 이고 분산이 1 인 표준 정규 분포로 변환, Z-Score**
- 특성의 최솟값과 최댓값 크기를 제한하지는 않음
- 이상치(outlier)를 파악가능

$$z = \frac{\mathrm{X} - mean(x)}{stdev(x)}$$

데이터 응용

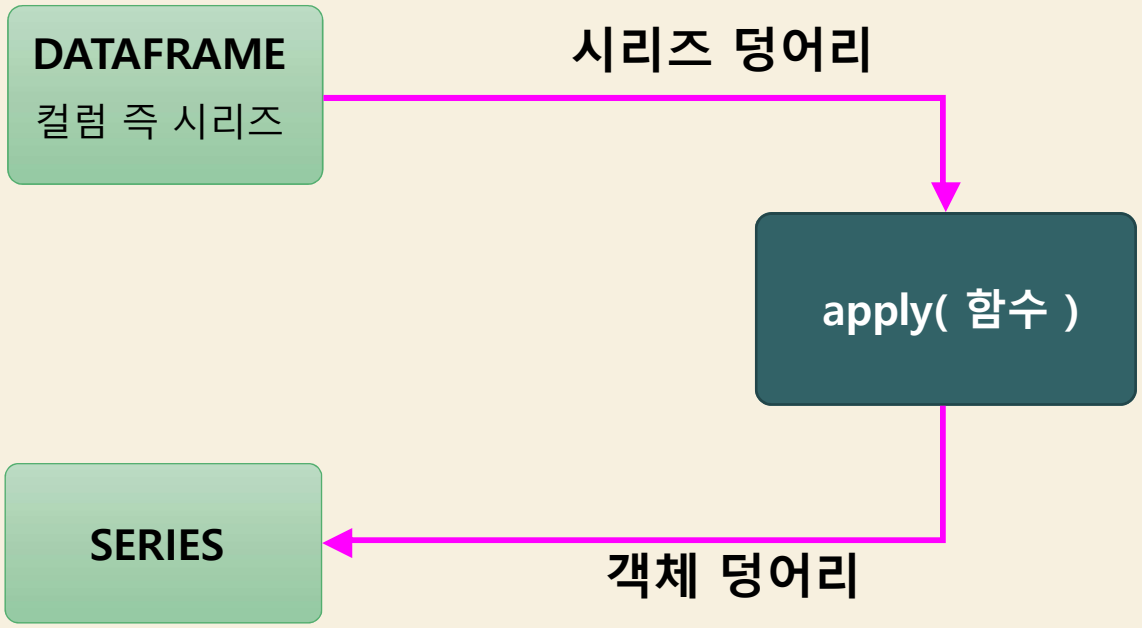
PANDAS 데이터 응용

◆ 함수 매핑



PANDAS 데이터 응용

◆ 함수 매핑



PANDAS 데이터 응용

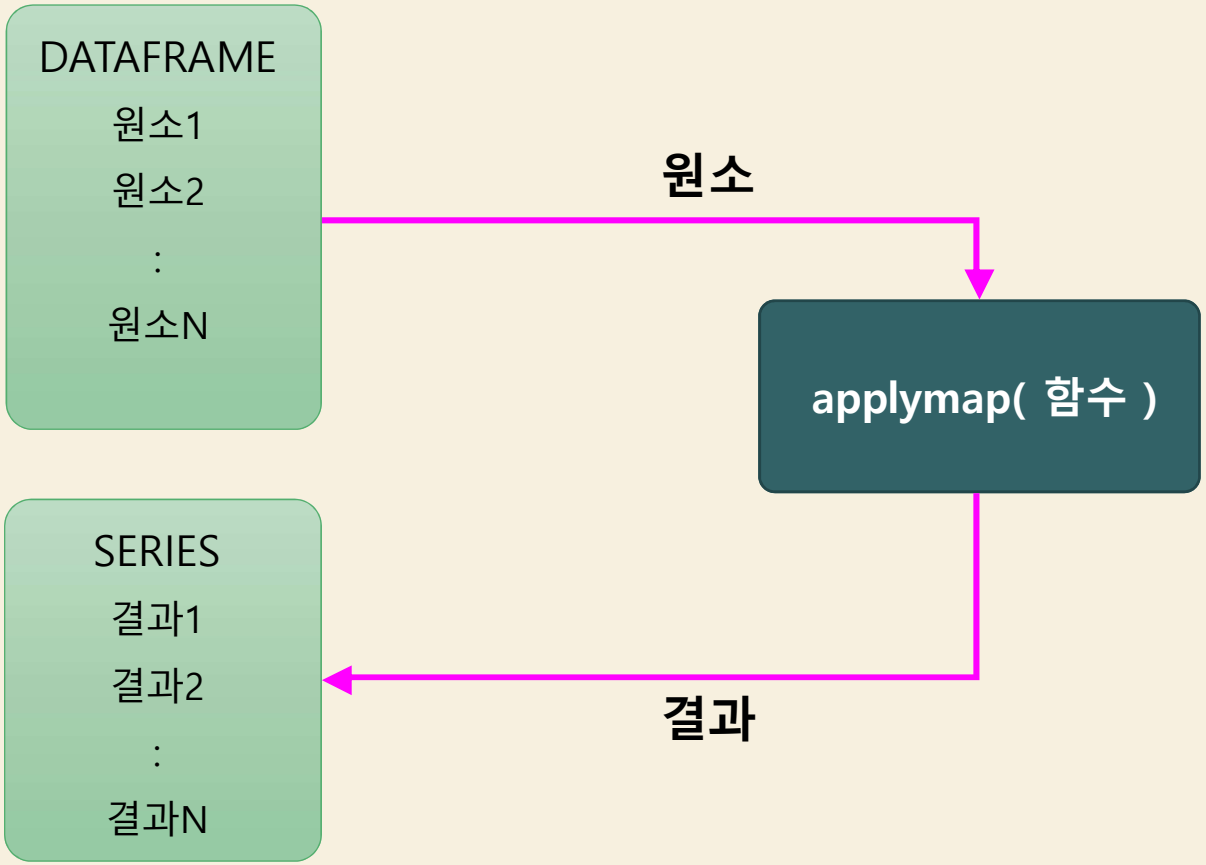
◆ 함수 매핑

```
object.apply( 함수명,          # 내가 만든 함수
              args=( ),        # 객체 이외에 함수에 전달되는 인자
              **kwargs,        # 객체 이외에 함수에 전달되는 가변인자
              axix=0,          # 0 or 'index': apply function to each column.
                               #1 or 'columns': apply function to each row.
              result_type=     # expand - 열에 리스트와 유사한 결과 반환
                               # reduce - 시리즈 반환,
                               # broadcast - 데이터프레임
             )
```

반환 : 단일 값, 시리즈, 데이터프레임

PANDAS 데이터 응용

◆ 함수 매핑



PANDAS 데이터 응용

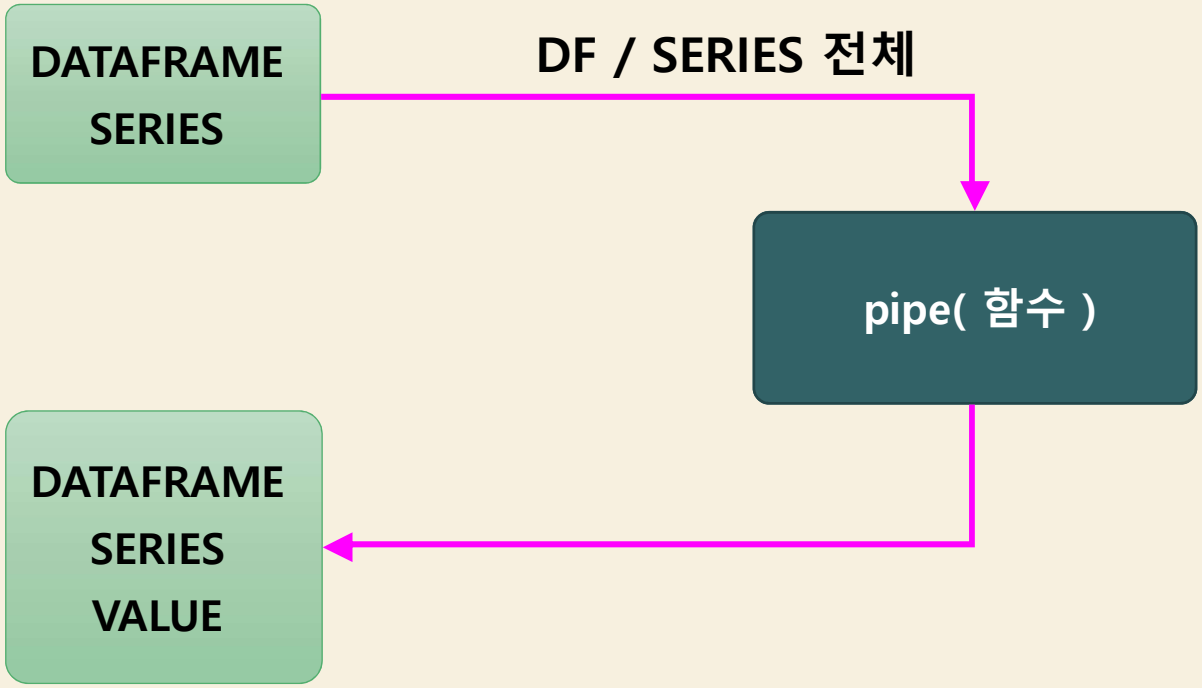
◆ 함수 매핑

```
object.applymap( func,
                 na_action,   # NaN 데이터 처리
                 **kwargs     # 객체 이외에 함수에 전달되는 가변인자
                 )
```

반환 : DataFrame

PANDAS 데이터 응용

◆ 함수 매핑



PANDAS 데이터 응용

◆ 함수 매핑

```
object.pipe (    func,
                      args=( ),          # 객체 이외에 함수에 전달되는 인자
                     **kwargs,          # 객체 이외에 함수에 전달되는 가변인자
                      axis=0
                )
```

반환 : 단일값, 시리즈, 데이터프레임

PANDAS 데이터 응용

◆ 열 재구성 - 열 순서 변경

정렬된 리스트 = sorted(column name , reverse)

반복가능한 객체 = reversed(column name)

PANDAS 데이터 응용

◆ 열 재구성 - 열 분리

분리된 문자열 리스트 = Series.str.split(구분자)

분리된 데이터= Series.str.get(인덱스)

분리된 문자열 리스트 = DataFrame.__str__.split(구분자)

분리된 데이터= ListObj.__getitem__(인덱스)

PANDAS 데이터 응용

◆ 필터링(Filtering)

- Series, DataFrame에서 **특정 조건식 만족하는 원소만 추출**
- 대표적인 방법 ➔ **불린 인덱싱(Boolean Indexing)**

PANDAS 데이터 응용

◆ 필터링(Filtering)

❖ 불린 인덱싱(Boolean Indexing)

- Series 객체 ➔ 비교 연산자 + 논리 연산자 ➔ True/False 결과

False					
False					
True					
False					
True					
True					
False					

PANDAS 데이터 응용

◆ 필터링(Filtering)

❖ 불린 인덱싱(Boolean Indexing)

- 비교 연산자 ➔ >, >=, <, <=, ==, !=
- 논리 연산자 ➔ (왼쪽 결과) & (오른쪽 결과)

  (왼쪽 결과) | (오른쪽 결과)

  ~ (조건식)

PANDAS 데이터 응용

◆ 필터링(Filtering)

❖ 특정 값 가진 행 추출

```
DataFrame[열].isin( 추출값 리스트 )
```

PANDAS 데이터 응용

◆ 데이터프레임 합치기

❖ 연결

구성 형태와 속성이 균일하다면 행 또는 열 중에
어느 한 방향으로 이어 붙이는 것 ➔ 일관성 유지

```
pandas.concat( 데이터프레임 리스트,
               axis=0,                  # 행 방향
               ignore_index=False,      # 행 인덱스 유지
               join='outer',            # 열이름 합집합
               kyes=[]                  # 행/열 이름 재정의
               )
```

PANDAS 데이터 응용

◆ 데이터프레임 합치기

❖ 연결

- **result1 = pd.concat([df1, df2])**

행 방향 , 인덱스 유지, 열이름 합집합

4 x 3

	a	b	c
0	a0	b0	c0
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c3

4 x 4

	a	b	c	d
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5

8 x 4

	a	b	c	d
0	a0	b0	c0	NaN
1	a1	b1	c1	NaN
2	a2	b2	c2	NaN
3	a3	b3	c3	NaN
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5

PANDAS 데이터 응용

◆ 데이터프레임 합치기

❖ 연결

- **result1 = pd.concat([df1, df2], join='inner')**

행 방향 , 인덱스 유지, 열이름 교집합

4 x 3

	a	b	c
0	a0	b0	c0
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c3

4 x 4

	a	b	c	d
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5

8 x 3

	a	b	c
0	a0	b0	c0
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c3
2	a2	b2	c2
3	a3	b3	c3
4	a4	b4	c4
5	a5	b5	c5

PANDAS 데이터 응용

◆ 데이터프레임 합치기

❖ 연결

- **result1 = pd.concat([df1, df2], ignore_index=True)**
- 행 방향 , 인덱스 변경, 합집합

4 x 3

	a	b	c
0	a0	b0	c0
1	a1	b1	c1
2	a2	b2	c2
3	a3	b3	c3

4 x 4

	a	b	c	d
2	a2	b2	c2	d2
3	a3	b3	c3	d3
4	a4	b4	c4	d4
5	a5	b5	c5	d5

8 x 4

	a	b	c	d
0	a0	b0	c0	NaN
1	a1	b1	c1	NaN
2	a2	b2	c2	NaN
3	a3	b3	c3	NaN
4	a2	b2	c2	d2
5	a3	b3	c3	d3
6	a4	b4	c4	d4
7	a5	b5	c5	d5

PANDAS 데이터 응용

◆ 데이터프레임 합치기

❖ 연결

DataFrame의 append 메서드

```
dataframe.append( 데이터프레임 리스트,
                  sort=False,                 # 열 정렬
                  ignore_index=False,         # 행 인덱스 유지
                  verify_integrity=False)     # 중복인덱스 검사
```

PANDAS 데이터 응용

◆ 데이터프레임 합치기

❖ 병합

어떤 기준에 의해 두 DataFrame을 병합(합치)하는 것

기준이 되는 열이나 인덱스를 **Key**라함

```
pandas.merge( df_left,
              df_right,
              how='inner',     # 기준 열 양쪽 모두 존재 교집합
              on=None,         # None : 공통 모든 열 키로 설정
              left_on=왼쪽 키,
              right_on=오른쪽 키
            )
```

PANDAS 데이터 응용

◆ 데이터프레임 합치기

❖ 결합

merge()함수를 기반으로 만들어진 함수

단, 행 인덱스 기준 결합

```
DataFrame.join( df_left,
                df_right,
                how='',    # 기준 열 양쪽 모두 존재 교집합
                on=None,       # 공통 모든 열 키로 설정
                left_on=왼쪽 키,
                right_on=오른쪽 키
            )
```

PANDAS 데이터 응용

◆ 그룹연산

❖ 특정 기준 적용하여 몇 개의 그룹으로 분할하여 처리하는 것

❖ 데이터 집계, 변환, 필터링에 효율적

[프로세스]

① 분할(Split) : 데이터를 특정 조건에 의해 분할

② 적용(apply) : 데이터 집계, 변환, 필터링에 필요한 메서드 적용

③ 결합(Combine) : 2단계의 처리 결과 하나로 결합

PANDAS 데이터 응용

◆ 그룹연산

❖ GroupBy Object

- DataFrame.**groupby()**, Series.**groupby()** 등의 메서드에 의해서 반환되는 반복이 가능한 객체
- 그룹 키와 데이터로 구성 ➔ **Dict 형태**
- 다양한 연산 메서드 사용

		age	fare	survived
class	sex			
First	female	34.611765	106.125798	0.968085
	male	41.281386	67.226127	0.368852
Second	female	28.722973	21.970121	0.921053
	male	30.740707	19.741782	0.157407
Third	female	21.750000	16.118810	0.500000

PANDAS 데이터 응용

◆ 그룹연산

❖ GroupBy Object Method

`GroupBy.__iter__()`	Groupby iterator.
`GroupBy.groups`	Dict {group name -> group labels}.
`GroupBy.indices`	Dict {group name -> group indices}.
`GroupBy.get_group`(name[, obj])	Construct DataFrame from group with provided name.
`Grouper`(*args, **kwargs)	A Grouper allows the user to specify a groupby instruction for an object.

PANDAS 데이터 응용

◆ 그룹연산

❖ 집계(Aggregation)

- 그룹객체에 다양한 연산 적용하는 것 ➔ 데이터 집계
- GroupBy 객체의 mean(), sum(), max(), min(), count(), size(), var(), std(), describe(), info(), last(), agg() 등 메스드
- 매핑함수 ➔ **agg(함수)**

  그룹별 연산 가능 컬럼 데이터만 연산 실행

PANDAS 데이터 응용

◆ 그룹연산

❖ 변환(Transformation)

- 그룹별로 구분하여 각 원소에 함수 적용
- 각 원소 원래 행인덱스, 열이름 기준으로 결과 반환
- 매핑함수 ➔ **transform(함수)**

  그룹별 연산 가능 컬럼 데이터만 연산 실행

PANDAS 데이터 응용

◆ 그룹연산

❖ 변환(Transformation)

[Z-score]

- 데이터 분포 중심에서 멀리 떨어진 데이터 즉 이상치를 찾는 방법 중 하나
- 평균과 표준오차에서 얼마나 벗어나 있는지 측정

$$Z - score = \frac{x_i - \mu}{\sigma}$$

PANDAS 데이터 응용

◆ 멀티인덱스

level 0 / level 1

class	sex	age	fare	survived
First	female	34.611765	106.125798	0.968085
	male	41.281386	67.226127	0.368852
Second	female	28.722973	21.970121	0.921053
	male	30.740707	19.741782	0.157407
Third	female	21.750000	16.118810	0.500000

PANDAS 데이터 응용

◆ 멀티인덱스

❖ 인덱스 레벨별 데이터 추출 인텍서 ➔ xs()

- 값 설정용으로는 불가

```
object.xs( key,
           axis=0,
           level,          # 숫자 또는 레벨명
          drop_level
          )
```

PANDAS 데이터 응용

◆ 피벗(Pivot)

- ❖ 많은 양 데이터에서 **필요한 자료만 뽑아 새롭게 표 생성**
- ❖ **사용자 임의대로** 데이터를 정렬하고 필터링 가능
- ❖ 구성요소
  - ▪ 행 인텍스
  - ▪ 열 인덱스
  - ▪ 데이터 값
  - ▪ 데이터 집계 함수

PANDAS 데이터 응용

◆ 피벗(Pivot)

```
pd.pivot_table(  df,                  # 피벗할 데이터프레임
                 index=' ',           # 행 위치에 들어갈 열
                 columns=' ',         # 열 위치에 들어갈 열
                 values='',           # 데이터로 사용할 열
                 aggfunc=''           # 데이터 집계 함수
               )
```

CH4 – 데이터 시각화

PANDAS 내장 그래프

◆ 내장 그래프

>> Matplotlib 라이브러리 *일부 기능 내장*

>> 간단한 그래프 출력

>> 데이터 시각화 다양한 처리는 시각화 라이브러리로 처리 필요

PANDAS 내장 그래프

◆ 내장 그래프

함수 ➔ 객체.plot()

kind 옵션	설명
line	선 그래프 (기본값)
bar	수직 막대 그래프
barh	수평 막대 그래프
his	히스토그램
box	박스 플롯

kind 옵션	설명
kde	커널 밀도 그래프
area	면적 그래프
pie	파이 그래프
scatter	선점도 그래프
hexbin	고밀도 선전도 그래프

PANDAS 내장 그래프

◆ 내장 그래프

그래프	메서드	
line	Series.plot()	DataFrame.plot()
bar	Series.plot(kind='bar')	DataFrame.plot(kind='bar')
barh	Series.plot(kind='barh')	DataFrame.plot(kind='barh')
his	Series.plot(kind='his')	DataFrame.plot(kind='his')
box	Series.plot(kind='box')	DataFrame.plot(kind='box')
pie	Series.plot(kind='pie')	DataFrame.plot(kind='pie')
area	Series.plot(kind='area')	DataFrame.plot(kind='area')
scatter	Series.plot(kind='scatter')	DataFrame.plot(kind='scatter')

PANDAS 내장 그래프

◆ 내장 그래프

```
# 모듈 로딩 ----------------------------------------------------------
import pandas as pd
import matplotlib.pyplot as plt

# 데이터프레임 데이터  준비 ------------------------------------------
data = { 'a' : [2, 1, 2, 9] , 'b' : [1, 0, 1, 2] }
df = pd.DataFrame ( data , index=[ -1, 0, 1, 2 ])
print(' Data ======₩n', df)

# 데이터 시각화 -----------------------------------------------------
df.plot()                    # 라인 플롯(선그래프) 생성
#df.plot(kind='line')
#df.plot.line()

plt.show()                                  # 플롯(그래프) 출력
```

PANDAS 내장 그래프

◆ 내장 그래프

- 막대 그래프

>> 데이터 시각화에 가장 많이 사용되는 그래프
>> 범주형 데이터 시각화에 사용

- 선 그래프

>> 시간적 순서 가진 데이터 표현에 사용되는 그래프
>> 순차적이고 연속적인 값 표현

- 히스토그램

>> 막대 그래프와 유사한 형태
>> 연속형 데이터의 값 분포 시각화에 사용
>> 변수 분포, 중심 경향, 퍼짐 정도, 치우침 정도 등 한눈에 확인
>> 구간 기본값 : 10

PANDAS 내장 그래프

◆ 내장 그래프

- 산점도 그래프

>> 2개 이상 열(column) 관계 분석 그래프
>> 값 분포 확인 가능

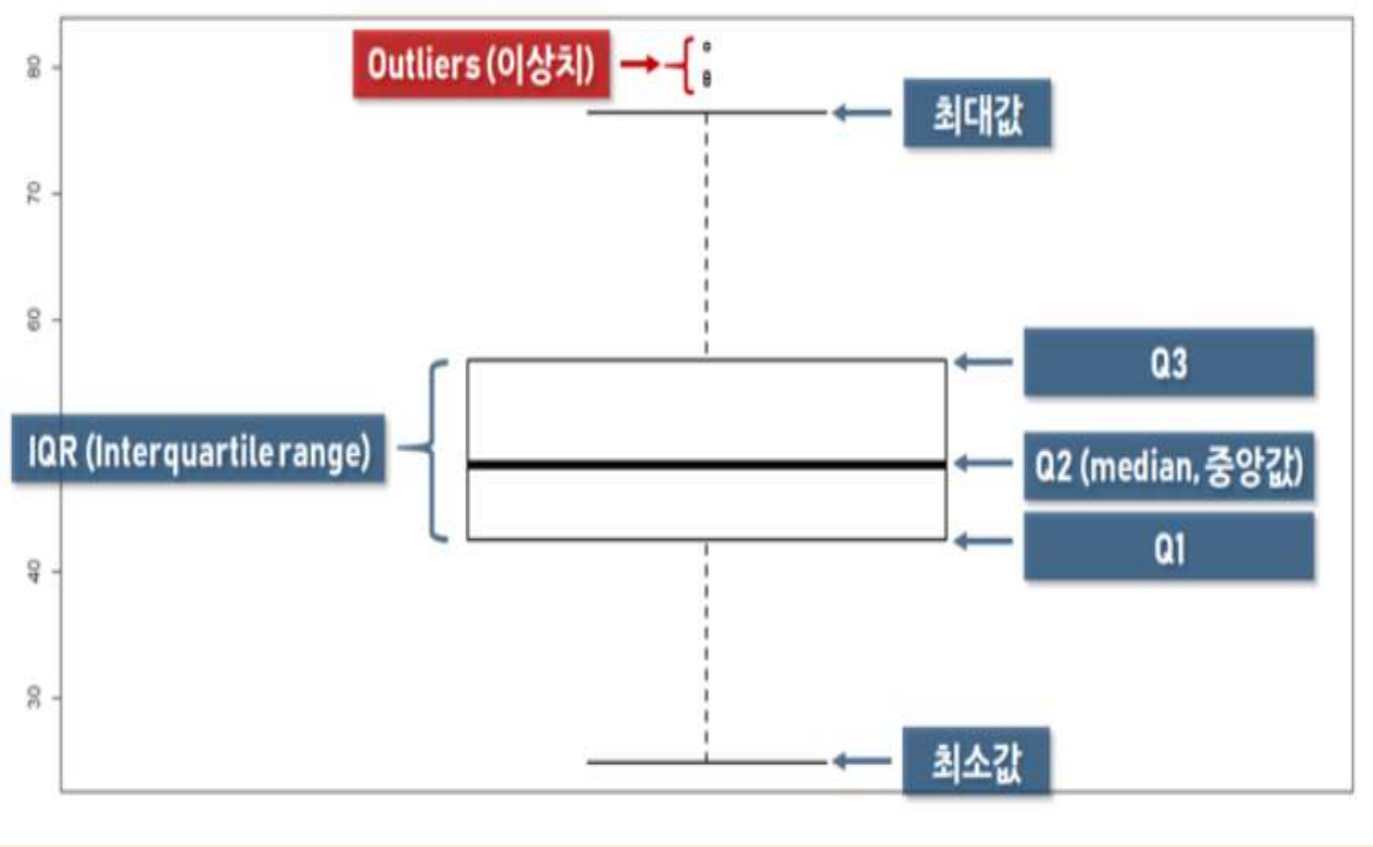
- 박스 플롯

>> 특정 변수의 데이터 분포, 분산 정도 정보 시각화
>> 한 눈에 데이터 파악 가능

PANDAS 내장 그래프

◆ 내장 그래프

- 박스 플롯



MATPLOTLIB 시각화

◆ Matplitlib 모듈

>> 챠트나 플롯으로 시각화(Visulaization)하는 모듈

>> 저수준 API 사용한 다양한 시각화 기능 제공

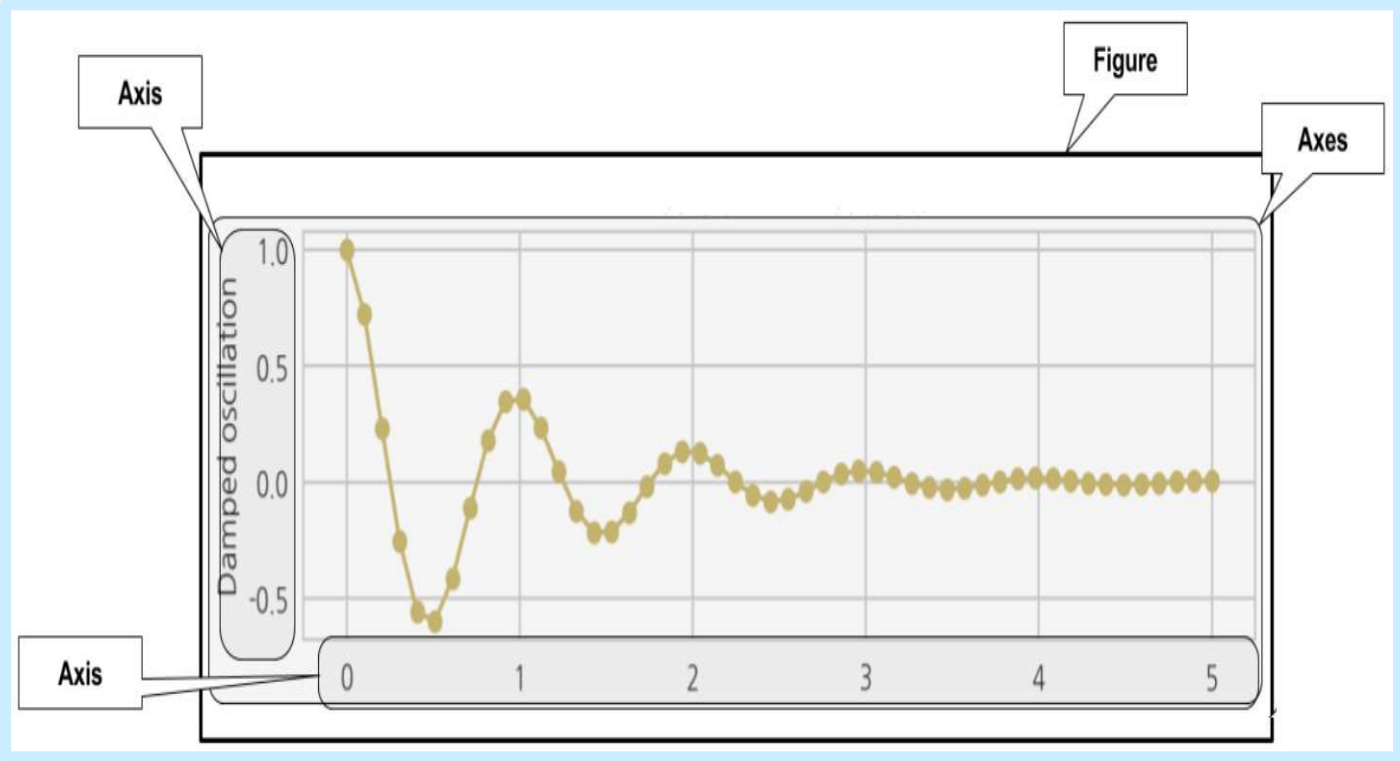
>> 서브 패키지 **pyplot** 활용

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

MATPLOTLIB 시각화

◆ Matplitlib 모듈

- 구성 ➔ Figure 객체 , Axes 객체, Axis 객체



MATPLOTLIB 시각화

◆ Matplitlib 모듈

- 구성요소

[Figure 객체]

- Matplotlib가 그리는 그림 객체
- 하나의 아이디와 윈도우 생성
- **plot()으로 자동 생성**
- Matplotlib.figure.Figure
- https://matplotlib.org/api/figure_api.html#Matplotlib.figure.Figure

[Axes 객체]

- 하나의 Figure 안에 여러 개의 Plot을 배열형태로 보이는 경우 사용
- **subplot(행, 열, index)로 생성**
- plot()으로 자동 생성
- Matplotlib.figure.Figure
- https://matplotlib.org/api/figure_api.html#Matplotlib.figure.Figure

MATPLOTLIB 시각화

◆ Matplitlib 모듈

- 스타일
  - 다양한 스타일 출력 지원

```
import matplotlib.pyplot as plt


# 스타일 리스트 출력
styles=plt.style.available
for style in styles:
    print(style)
```

적용 ➔ plt.style.use('ggplot')

```
Solarize_Light2
_classic_test_patch
bmh
classic
dark_background
fast
fivethirtyeight
ggplot
grayscale
seaborn
seaborn-bright
seaborn-colorblind
seaborn-dark
seaborn-dark-palette
seaborn-darkgrid
```

MATPLOTLIB 시각화

◆ Matplitlib 모듈

- 한글폰트 설정

```
from matplotlib import font_manager as fm, rc

# 한글 폰트 설정
font_path=r'C:₩Windows₩Fonts₩malgun.ttf'
font_path='.₩malgun.ttf '

font_name=fm.FontProperties(fname=font_path).get_name()
rc('font', family=font_name)
```

MATPLOTLIB 시각화

◆ Matplitlib 모듈

- Line Plot

```
plot(*args,                    # [x], y, [frmt],  [x2], y2, [fmt2]
        scalex=True,
        scaley=True,
        data=None,
        **kwargs)              #  선 레이블(자동 범례의 경우), 선폭, 앤티앨리어싱,
                                  마커 면 색상과 같은 속성 지정
```

```
>>> plot(x, y)        # plot x and y using default line style and color
>>> plot(x, y, 'bo')  # plot x and y using blue circle markers
>>> plot(y)           # plot y using x as index array 0..N-1
>>> plot(y, 'r+')     # ditto, but with red plusses
```

MATPLOTLIB 시각화

◆ Matplitlib 모듈

- 그래프 생성 및 표시 -- 기본 Line 그래프

```
# 기본 그래프 생성
plt.plot( ['a','c','e'], [3,5,7])        # 데이터 plot(x축 데이터, y축 데이터)

plt.plot( [1,2,3,4] )                    # 데이터 plot(y축 데이터)  x= 인덱스

# 그래프 화면 표시
plt.show()                               # 화면 출력 & 마우스 이벤트 대기
```

MATPLOTLIB 시각화

◆ Matplitlib 모듈

- 그래프 꾸미기

타이틀 & x/y축 라벨

```
plt.title( " 그래프 타이틀 ", size=n )

plt.xlabel( " x축 라벨", size=n )
plt.ylabel( " y축 라벨", size=n )
```

MATPLOTLIB 시각화

◆ Matplitlib 모듈

- 그래프 꾸미기

x축 라벨 간격 조절

```
# 그림 사이즈 지정(가로 14인치, 세로 5인치)
plt.figure( figsize=(14, 5) )

# x축 눈금 라벨 회전
plt.xticks( rotation='vertical' )
```

← vertical, horizontal

← 45, 90, 180 각도

MATPLOTLIB 시각화

◆ Matplitlib 모듈

- 그래프 꾸미기

범례 설정

plt.legend(labels=[' 범례 '], loc= ' 위치')

Location String	Location Code
'best'	0
'upper right'	1
'upper left'	2
'lower left'	3
'lower right'	4
'right'	5
'center left'	6
'center right'	7
'lower center'	8
'upper center'	9
'center'	10

MATPLOTLIB 시각화

◆ Matplitlib 모듈

- 그래프 꾸미기

x/y축 범위 지정

```
# x, y축 범위 지정(최소값, 최대값)
plt.xlim( 최소값, 최대값)
plt.ylim( 최소값, 최대값)
```

MATPLOTLIB 시각화

◆ Matplitlib 모듈

- 그래프 꾸미기

라인 형태 포맷

plt.plot([x], y, [frmt])

Line2D에 상세속성

marker	'o', 'v', '^', '<', '>', '8', 's', 'p', '*', 'h', 'H', 'D', 'd', 'P', 'X
markersize	float n.n
markerfacecolor	마커 색상
linesize	'-' or 'solid' '--' or 'dashed' '-.' or 'dashdot' ':' or 'dotted' 'none', 'None', ' ', or ''
linewidth	float n.n--

SEABORN 시각화

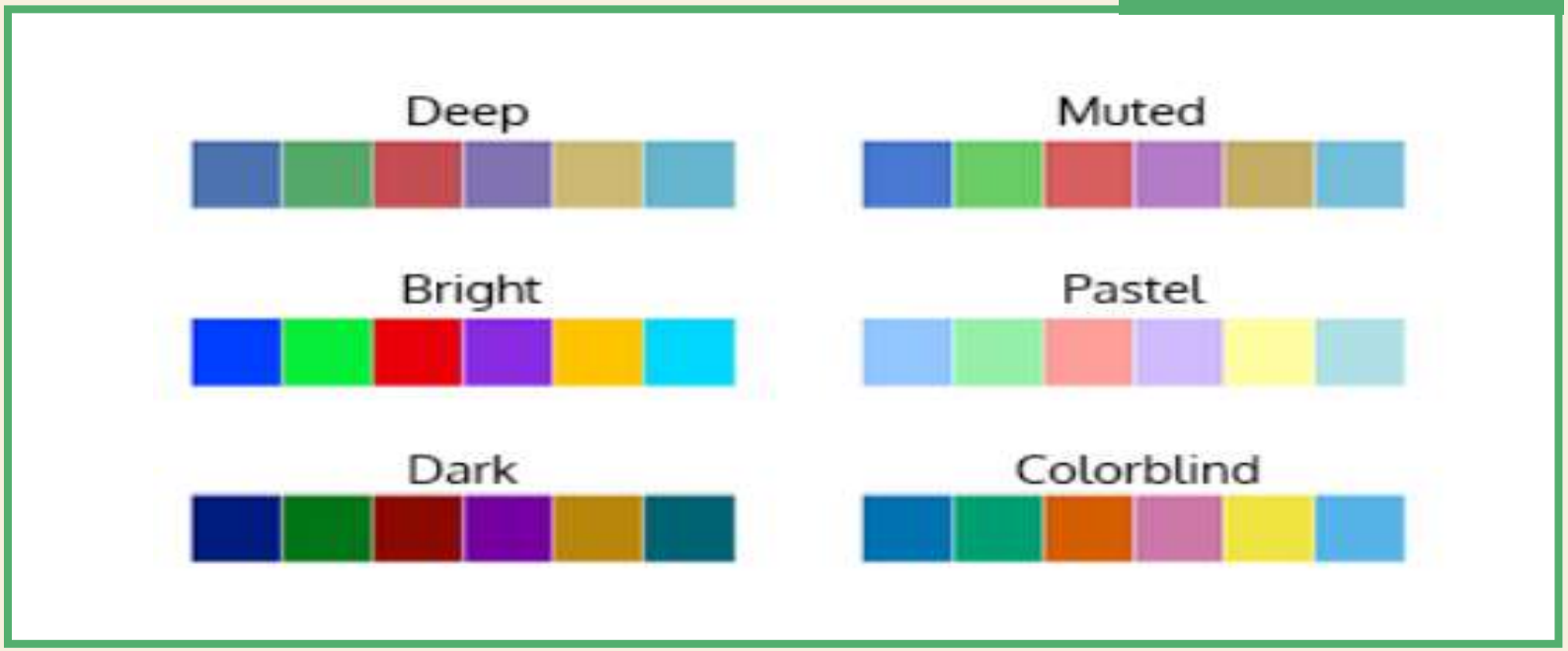
◆ Seaborn 라이브러리

- Matplotlib 기반 **다양한 색상 테마와 통계용 차트** 등 기능 추가함 시각화 패키지
- Matplotlib에 의존하며 통계 기능은 Statsmodels 패키지에 의존
- http://seaborn.pydata.org/

SEABORN 시각화

◆ Seaborn Color Palette

기본 팔렛트



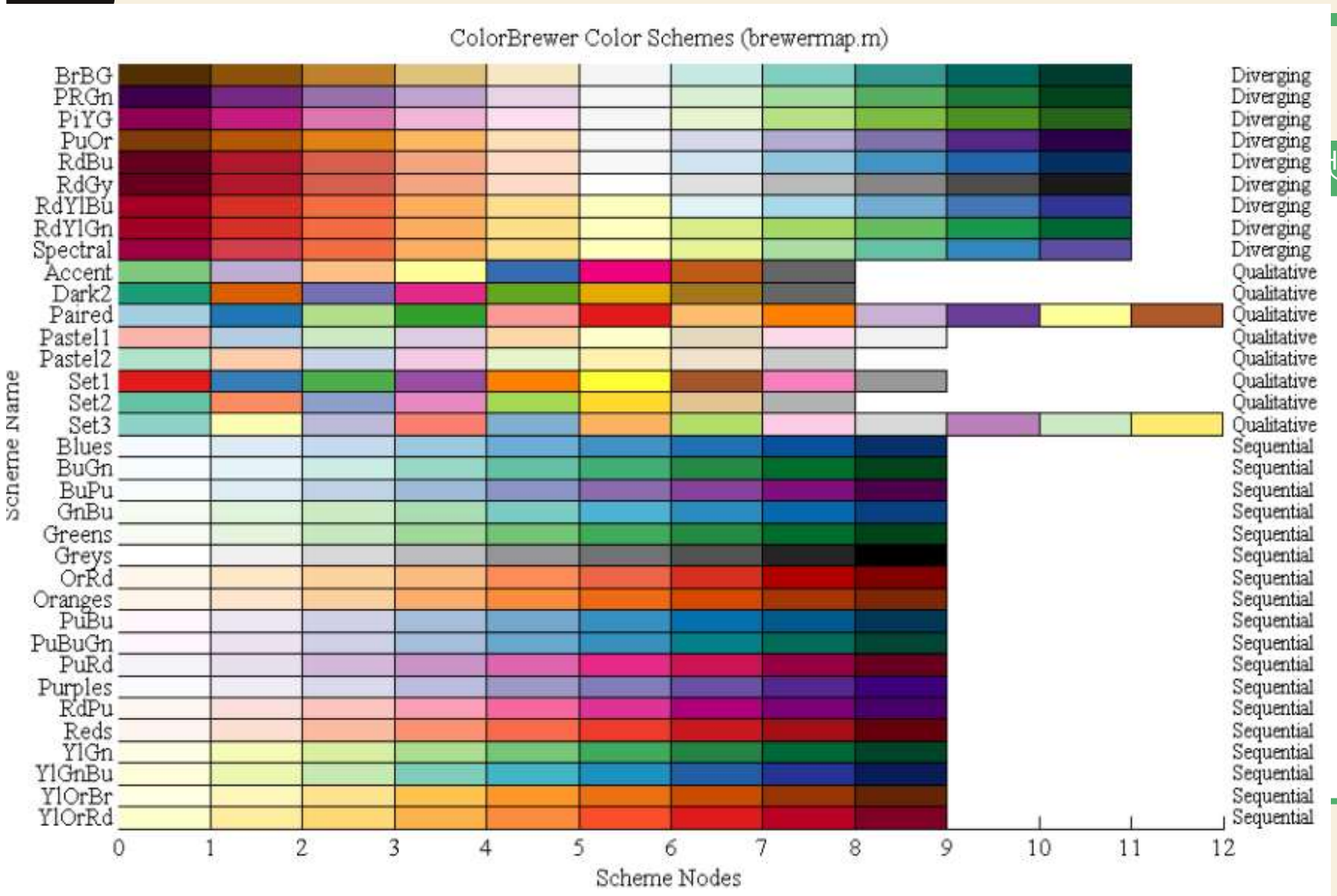
SEABORN 시각화

◆ Seaborn Color Palette

Color Brewer 팔렛트

- 지도 제작자 Cindy Brewer 연구에서 영감을 받아 만들어진 팔렛트 세트
- 카테고리에 따른 색상 팔레트
  - Qualitative : 범주형 자료형에 좋음,
  - Sequential : 밝은색 >> 어두운색 연속적, 등급 데이터에 좋음
  - Diverging : 양쪽 강조, 낮은값 - 높은 값 데이터에 좋음

SEABORN 시각화



SEABORN 시각화

◆ Seaborn 한글 폰트 설정

```
import seaborn as sns


# 폰트, 마이너스 폰트 설정, 스타일 설정
sns.set( font="AppleGothic",
         rc={"axes.unicode_minus":False},
         style='darkgrid' )
```

SEABORN 시각화

◆ Seaborn Level

Figure-Level Function

seaborn베이스로 figure 만들어서 plotting하며 FacetGrid 반환

Relational	relplot()	**Categorical**	catplot()
Distribution	distplot()	**Regression**	lmplot()
Matrix	ClusterGrid()	**Multi-plot**	FacetGrid() PairGrid() JointGrid()

Axes-Level Function

matplotlib 베이스로 matplotlib.axes.Axes를 반환하는 함수

- Figure-Level plot 함수 제외 나머지

SEABORN 시각화

◆ Seaborn 모듈

- 서로 **다른 데이터 관계 시각화** 산점도

서로 다른 2개 연속변수 사이 관계를 나타내는 그래프

2개 관계를 일차원 함수로 표현한 직선 ➔ 회귀분석(Regression)

```
sns.regplot(      x='age',          # x축 변수
                  y='fare',         # y축 변수
                  data=titanic,     # 데이터
                  ax=ax2,           # axe 객체 - 2번째 그래프
                  fit_reg=False)    # 회귀선 미표시
```

SEABORN 시각화

◆ Seaborn 모듈

- 단 변수 **데이터의 분포** 시각화 그래프

1개 변수 데어터의 분포 확인하는 것, 3가지 종류 그래프로 분석

hist (히스토그램 Histogram) : 수치형 데이터 분포 표현

kde(커널밀도추정 Kernel Density Estimator) : 히스토그램 대안

ecdf(경험적누적분포 Empirical cumulative distibutions) : 각 지점 데이터 모두 나타냄, y축은 x축의 값이 전체 데이터에서 차지하는 비율 의미

SEABORN 시각화

◆ Seaborn 모듈

- 단 변수 **데이터의 분포** 시각화 그래프

```
sns.distplot(       titanic['fare'],
                    ax=ax1,
                    kind='kde',          # 분석 방법 hist, kde
                    rug=False,
                    hist=True.
                    kde=True,

                )
```

SEABORN 시각화

◆ Seaborn 모듈

- 2개 **범주형 데이터 분석** ➔ 히트맵(hneatmap)

> - 2개의 범주형 변수를 각각 x, y축에 넣고 데이터를 매트릭스 형태로 분류한 상태
> - 데이터프레임을 피벗테이블로 만든 객체를 그림

```
sns.heatmap(table,                  # 데이터프레임
            annot=True,             # 데이터 값 표시 여부
            fmt='d',                # 정수형 포맷
            cmap='YlGnBu',          # 컬러 맵
            linewidth=.5,           # 구분 선
            cbar=False)             # 컬러 바 표시 여부
```

SEABORN 시각화

◆ Seaborn 모듈

- 2개 **범주형 데이터 분석** ➔ 산점도

```
sns.scatterplot(  x="class",
                  y="age",
                  data=titanic,
                  ax=ax3)
```

```
sns.stripplot(    x="class",      #x축 변수
                  y="age",        #y축 변수
                  data=titanic,   #데이터셋 - 데이터프레임
                  ax=ax1)         #axe 객체 - 1번째 그래프
```

SEABORN 시각화

◆ Seaborn 모듈

- 2개 **범주형 데이터 분석** ➔ 산점도

```
sns.swarmplot(   x="class",      #x축 변수
                  y="age",       #y축 변수
                 data=titanic,   #데이터셋 - 데이터프레임
                 ax=ax2)         #axe 객체 - 2번째 그래프
```

FOLIUM 지도 시각화

◆ Folium 패키지

- 지도 위에 데이터를 표현해주는 **지도 시각화 라이브러리**
- leaflet.js기반
- 지도 생성 후 마커 추가, 범위 표기 후 HTML 파일로 내보내기
- https://python-visualization.github.io/folium/

- 설치

```
pip install folium          # 터미널
!pip install folium         # jupyter notebook
```

FOLIUM 지도 시각화

◆ Folium 패키지

기본 좌표 설정

```
# 위치 = [ 위도, 경도 ]
location = [ latitude,  longitude ]


# 확대 정보
zoom_start = 최대 18
```

FOLIUM 지도 시각화

◆ Folium 패키지

마커 설정

```
# 위치 = [ 위도, 경도 ]
location = [ latitude,  longitude ]


# 확대 정보
zoom_start = 최대 18
```