ADP 실기 코드북

with Python

ver.01

목 차

Part 1. 데이터 핸들링

이 장은 데이터의 기본적인 조작을 위한 기법들을 소개하는 장입니다.
주로 pandas의 함수를 사용하며 데이터를 조작하는 방법을 소개합니다.

1. DataFrame 기본

✿ Pandas DataFrame 사용 준비

```
import pandas as pd          # 앞으로 pandas의 함수를 호출할 때 pd라는 별칭을 사용
```

✿ DataFrame 읽기

```
df = pd.read_csv(filepath, na_values='NA', encoding='utf8')
```

- ❖ filepath : 파일 경로 ('p1/p2/file.csv', 'p1₩₩p2₩₩file.csv' 또는 r'p1₩p2₩file.csv' 형태로 연결)
- ❖ na_values : csv 파일에서 null 값이 어떤 string 값으로 저장된 경우 사용
- ❖ encoding : 한국어 데이터인 경우 'utf8' 또는 'cp949', 'euc-kr' 형식으로 되어 있음

✿ DataFrame 저장

```
df.to_csv(filepath, header=True, index=True, encoding='utf8')
```

✿ DataFrame 선언

```
df = pd.DataFrame(data)
```

✿ DataFrame 출력

```
df.head(n)    # 처음 n개 출력, 미지정시 5개
df.tail(n)    # 마지막 n개 출력, 미지정시 5개
```

✿ DataFrame 전체 출력하기 옵션 지정

```
pd.set_option( 'display.max_columns', None )   # 모든 열 출력
pd.set_option( 'display.max_rows', None )   # 모든 행 출력
```

✿ DataFrame 요약 / 통계정보 확인

```
df.info( )   # 총 데이터 건수와 데이터 타입, Null건수 확인
df.describe( )   # 숫자형 컬럼들의 n-percentile분포도, 평균값, 최댓값, 최솟값 확인


df[ '컬럼명' ].min( ) 또는 df[ '컬럼명' ].max( )    # 해당 컬럼의 min, max 값을 확인
```

✿ DataFrame 인덱스 확인 / 추가 / 리셋

DataFrame의 index를 출력하는 함수로, 결과를 list() 함수로 묶어 변수로 할당하면 편리하게 사용할 수 있다.

```
df.index        # 인덱스 확인
```

DataFrame에 index를 추가하거나 reset하는 함수이다.

```
df.set_index(keys, drop = True, append = False, inplace = True)        # 인덱스 추가
df.reset_index(drop = False, inplace = Fales)     # 인덱스 리셋
```

✿ DataFrame 컬럼명 확인

```
df.columns
```

✿ DataFrame 컬럼명 변경

```
df.rename( { 'old1':'new1', 'old2':'new2' }, axis = 'columns' )

df.columns = [ 'new1', 'new2' ]

df.columns = df.columns.str.replace('기존문자', '대체할 문자')    # 컬럼명 특정문자 대체
```

✿ DataFrame 컬럼 타입 확인

```
df.dtypes
```

✿ DataFrame 컬럼 타입 변경

```
df['컬럼명'] = df['컬럼명'].astype('타입')


# 모든 컬럼에 대해 가능한 가장 적절한 dtypes로 변경

df = df.convert_dtypes( )
```

2. row/column 선택/추가/삭제

✿ row/column 선택

row의 index 또는 컬럼명을 사용하는 방법

```
df [ '컬럼명' ]          # 하나의 컬럼을 Series 형식으로 출력
df [[ '컬럼명1' , '컬럼명2' ]]          # 여러 개의 컬럼은 Data Frame 형식으로 출력
df.loc [[ 행 index, 컬럼명 list ]]     # 행과 열을 지정하여 출력
                                        모든 행 또는 모든 열을 출력할 때는 : 을 대신 사용
```

row 또는 column의 순서(위치)를 사용하는 방법

```
df.iloc [ 행 위치 list, 컬럼 위치 list ]
df.iloc [ n:m, N:M ]     # 이어진 행 또는 열을 출력할 때는 슬라이싱을 사용
```

row 또는 column의 순서(위치)를 사용하는 방법

```
df.iloc [ 행 위치 list, 컬럼 위치 list ]
df.iloc [ n:m, N:M ]     # 이어진 행 또는 열을 출력할 때는 슬라이싱을 사용
```

✿ column 추가 / 삭제

```
df [ '컬럼명' ] = 선언할 값          # 컬럼 추가
df.drop ( columns = [ '컬럼명1' , '컬럼명2' ], inplace = True )       # 컬럼 삭제
```

✿ row 추가 / 삭제

```
df.append (df와 컬럼명이 같은 데이터프레임)        # 행 추가
df.drop (인덱스 리스트)        # 행 삭제
```

데이터프레임에서 결측 데이터를 가진 행이나, 내용이 완전히 동일한 행을 삭제할 수 있음

```
df.dropna ( inplace = True )        # 결측치를 가진 행 삭제
df.drop_duplicates ( )        # 중복 행 삭제
```

3. 조건에 따른 row, column 선택 및 삭제

✿ 조건에 따른 column 선택

두 개 이상의 조건을 사용할 때 관계연산자의 우선순위를 모른다면 괄호를 많이 사용하는 것을 추천

```
df [df [ '컬럼명' ] >= int ]        # 조건이 하나일 때
df [ ( 조건1) & ( 조건1) & (( 조건1) | ( 조건1)) ]      # 조건이 두 개 이상일 때
```

✿ 조건에 따른 column 추가 / 변경

> '추가/변경할 컬럼명'이 현재 DataFrame에 존재하는 컬럼인 경우
> 조건문에 부합하는 행의 '추가/변경할 컬럼명'의 값을 변경함

> '추가/변경할 컬럼명'이 존재하지 않는 컬럼인 경우 새로운 컬럼을 만들고
> 조건문에 부합하는 행에는 지정된 값을, 그 외의 행에는 NaN을 삽입

```
df.loc [ 조건문, '추가/변경할 컬럼명' ] = 삽입할 값
```

✿ 조건이 여러 개일 때 column의 값 변경

조건이 여러개일 때 조건 목록과 선택 목록을 입력으로 받아서
조건에 따라 선택 목록의 요소로 구성된 배열을 반환

```
np.select ( 조건목록, 선택목록, default = 디폴트값 )


# 예시
import numpy as np
conditionlist = [ (baseball [ 'year' ] >= 2000),                                    # 조건
목록
                  (baseball [ 'year' ] >= 1950) & (baseball['year'] < 2000),
                  (baseball [ 'year' ] <= 1950) ]
choicelist = [ 'High', 'Mid', 'Low' ]                                                #
선택 목록
baseball [ 'year_lv' ] = np.select( conditionlist, choicelist, default='Not Specified' )
```

4. 데이터 정렬

✿ Index 기준 정렬

내림차순 정렬 : ascending = False로 지정 (default는 오름차순)

```
df.sort_index( )              # df 객체 행 인덱스 기준으로 정렬
df.sort_index( axis = 1 )     # df 객체 column 명으로 정렬
```

✿ 값 기준 정렬

내림차순 정렬 : ascending = False로 지정 (default는 오름차순)

```
df.sort_values( by= '컬럼명' )              # df 객체를 지정한 컬럼의 값 기준으로
정렬
df.sort_values( by= [ '컬럼명1', '컬럼명2' ] )     # 정렬 기준 컬럼이 여러 개일 때
```

5. 데이터 결합

결합하려는 데이터프레임의 행과 열의 개수가 맞지 않으면 NaN값이 채워짐

✿ row / column 결합

- 특정 축을 따라 pandas 객체(Series, DataFrame)를 연결
- ignore_index=True를 붙이면 인덱스 재배열 가능
- df-df 외에도 df-series, series-series 모두 가능

```
pd.concat([ df1, df2, ... ] ,          # 연결할 객체 리스트
          axis=0,                      # 0이면 행결합, 1이면 열결합
          join='outer',                # 인덱스가 모두에 존재하는 게 아닐 때 처리법
          ignore_index=False)      # 결합에 있어 인덱스를 무시할 지 여부
```

✿ key 기준 결합

on과 (left_on, right_on)은 함께 사용하지 않음
두 테이블에서 조인할 컬럼의 이름이 같다면 on, 다르다면 left_on과 right_on을 각각 지정

```
df_left.merge(df_right,
              on = None,              # 조인할 열 이름 (str or list)
              left_on = None, right_on = None,      # 각 df의 조인할 열 이름 (str or list)
              sort = False)           # 조인 키에 대한 정렬 여부
```

6. 데이터 요약

✿ 그룹화와 집계

```
df.groupby(by = None,          # 그룹을 결정하는데 사용
           axis = 0,           # 행(0)과 열(1) 지정, default 0
           level = None,       # 축이 계층 구조인 경우 특정 수준을 기준으로 그룹화
           as_index = True,    # 그룹 레이블이 인덱스로 출력될지의 여부, default True
           sort = False        # 집계 행으로 정렬할지 여부
           dropna = True)      # True이면 NA 값이 행/열과 함께 삭제, False이면 NA 값도 그룹의 키로 처리됨
```

- 대량의 데이터를 그룹화하고 이러한 그룹에 대한 작업을 계산
- df.groupby('그룹화할 기준열').FUN()['집계함수 적용시킬 열'] 형식으로 사용
- FUN() 자리에 min(), max(), count(), mean(), transform() 등을 주로 사용

✿ 기준이 하나일 때 분포 확인 : 도수분포표

각 계급에 속하는 자료의 개수인 도수를 조사하여 자료의 분표 상태를 살펴보는 표

```
np.unique(df ['기준열'], return_counts = True)

pd.Series(df ['기준열'] ).value_counts( )
```

✿ 기준이 두개일 때 상대도수 확인 : crosstab

행, 열 요인 기준 별로 빈도를 세어 교차표(contingency table) 를 만들어줌

```
pd.crosstab(index,                  # 열 위치에서 집계될 범주형 변수 (컬럼1)
            columns,                # 행 위치에서 집계될 범주형 변수 (컬럼2)
            dropna = True,          # 항목이 모두 NaN인 열을 제외할지 여부
            normalize = False)      # 'all'또는 True를 전달하면 모든 값에 대해 정규화
                                    'index'를 전달하면 각 행에 대해 정규화
                                    'columns'가 전달되면 각 열에 대해 정규화
```

7. 데이터 모양 변경

✿ 여러 개의 컬럼 ⇒ variable, value 컬럼

각 계급에 속하는 자료의 개수인 도수를 조사하여 자료의 분표 상태를 살펴보는 표

```
pd.melt (df,
         id_vars,          # variable, value의 내용으로 들어가지 않을 컬럼의 이름 리스트
         var_name,         # variable 변수의 이름으로 지정할 문자열 (선택)
         value_name)       # value 변수의 이름으로 지정할 문자열 (선택)
```

✿ 범주형, 값 데이터 ⇒ 범주 개수만큼의 컬럼

각 계급에 속하는 자료의 개수인 도수를 조사하여 자료의 분표 상태를 살펴보는 표

```
pd.pivot_table(df,
               index,          # index로 만들 컬럼명
               columns,        # 컬럼으로 변환할 범주형 변수
               values,         # 컬럼으로 변환될 범주형 변수의 값
               aggfunc,        # values에 적용할 집계함수
               margins)        # 소계, 총합계를 의미하는 all 행/열 추가
```

8. 데이터프레임에 함수 적용하기

✿ 행/열 방향으로 주어진 함수를 한 번에 적용 : apply

각 계급에 속하는 자료의 개수인 도수를 조사하여 자료의 분표 상태를 살펴보는 표

```
df.apply(np.max, axis= 1)
```

apply에 사용할 수 있는 numpy 함수

min : 최솟값
max : 최댓값
mean : 평균
median : 중앙값

prod : 곱
sum : 합계
std : 표준편차
var : 분산

✿ 데이터 내부 각 요소를 처리하기

- lambda는 간단한 함수 개념
- map(lambda)를 사용할 때에는 시리즈 형식이어야 함 .
- map은 원소 접근이어야 함 (시리즈) 1:1 매핑 대체 원소 -> 원소.func()

```
data [ 'col_name' ] = data [ 'col_name' ].map( lambda x : x.func(), na_action = 'ignore')
```

9. 문자열 데이터 변환하기

✿ 인덱싱

```
df [ 'col_name' ].str [ n : m ]
```

✿ 분할

“ ” : 첫번째 파라미터 문자열 값을 기준으로 문자열이 분할됨 (여기서는 띄어쓰기로 분할)

expand : 분할 문자열을 별도의 열로 확장

```
df [ 'col_name' ].str.split( " ", expand = True )
```

✿ 시작,끝,포함 글자 인식

```
df.str.startswith( "시작글자" )
df.str.endswith( "끝글자" )
df.str.contains( "포함글자" )
```

10. 날짜 데이터 핸들링

날짜와 시간을 다루기 위해서는 **datetime** 모듈을 이용한다.

✿ 현재 날짜와 시간 산출하기

```
datetime.today( )                          # 현재 날짜와 시간 산출
datetime.today().year/month/day/hour   # 현재 연도/월/일/시각 산출
```

✿ 문자열을 날짜 형식으로 변환하기

```
datetime.strptime('날짜 문자열', '포맷')
```

예시 : datetime.strptime('2021-12-25 00:00:00', '%Y-%m-%d %H:%M:%S')

✿ 날짜 데이터를 원하는 문자열 포맷으로 변환하기

```
datetime객체.strftime('포맷')


# 예시
time = datetime.today()
time.strptime('%Y-%m-%d %H:%M:%S')
```

strftime(), strptime() 메서드에 사용되는 서식

%d	0을 채운 10진수 표기로 날짜를 표시
%m	0을 채운 10진수 표기로 월을 표시
%y	0을 채운 10진수 표기로 2자리 년도
%Y	0을 채운 10진수 표기로 4자리 년도
%H	0을 채운 10진수 표기로 시간 (24시간 표기)
%I	0을 채운 10진수 표기로 시간 (12시간 표기)
%M	0을 채운 10진수 표기로 분
%S	0을 채운 10진수 표기로 초
%f	0을 채운 10진수 표기로 마이크로 초 (6자리)
%A	locale 요일
%a	locale 요일 (단축 표기)
%B	locale 월
%b	locale 월 (단축 표기)
%j	0을 채운 10진수 표기로 년 중 몇 번째 일인지 표시
%U	0을 채운 10진수 표기로 년중 몇 번째 주인지 표시 (일요일 시작 기준)
%W	0을 채운 10진수 표기로 년중 몇 번째 주인지 표시 (월요일 시작 기준)

✿ 날짜 데이터의 연산

두 날짜와 시간 사이의 차이를 계산할 때 사용함

```
datetime.timedelta(days=0, seconds=0, microseconds=0, milliseconds=0, minutes=0,
                   hours=0, weeks=0)
```

```
# 예시
time = datetime.today()
time + timedelta(days=100)
```

Part 2. 데이터 전처리

1. 데이터 분할 기법

✿ 단순 임의 추출 : numpy의 random.choice()

데이터를 training data와 test data로 분할할 때 가장 많이 사용하는 표본추출 방법

```
np.ramdom.choice( a,                  # 배열 : 정수값일 경우 arange(a)명령으로 배열 생성
                  size = None,        # 정수 : 샘플 숫자
                  replace = True,     # True이면 복원추출, False이면 비복원 추출
                  p = None)           # 배열 : 각 데이터가 선택될 수 있는 확률

# 예시 : np.random.choice(a, size=None, replace=True, p=[2,4,6]
모집단 내 각 원소별로 표본으로 뽑힐 확률 p를 알고 있거나 명시적으로 지정하고자 할 때 사용
첫 번째 값이 뽑힐 확률 20%, 두번째가 뽑힐 확률 40%, 세번째가 뽑힐 확률은 60%로 지정됨
```

✿ 단순 임의 추출 : pandas의 DataFrame.sample()

```
df.sample( n = None,                  # 반환할 항목 수 (frac과 함께 사용 불가)
           frac = None,               # 반환할 항목의 비율 (n과 함께 사용 불가)
           replace = False,           # True이면 복원추출, False이면 비복원 추출
           weight = None,             # '변수명' 형식 : 설정시 특정 컬럼 기준 가중치 부여
           random_state = None,       # 숫자값 : 값을 지정하여 난수생성을 반복할 수 있음
           ignore_index = False)      # 원본의 인덱스를 무시하고 새로 지정할지 여부
```

✿ 층화 임의 추출

모집단이 이질적인 몇 개의 계층으로 이루어져 있을 때 모든 계층으로부터 원소를 임의로 추출하여 각 계층을 고루 대표할 수 있도록 랜덤하게 표본을 추출하는 방법

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split( ... )
```

파라미터 예시

- X # 데이터 테이블
- test_size = 0.2 # 테스트 사이즈 비율
- shuffle = True # True이면 복원추출, False이면 비복원 추출
- stratify = 변수 # None이 아닌 경우 데이터는 지정한 변수를 기준으로 계층화 되어 해당 변수의 비율을 유지하도록 추출
- random_state = 1004 # 난수 // 임의의 번호 지정, 같은 번호라면 같은 값이 나옴

2. 데이터 정규화

방식 : Scaler import ⇒ Scaler 설정 ⇒ 정규화 방법 학습 : scaler.fit()
⇒ 정규화 적용 : scaler.transform()
일회성의 경우 fit_transform()을 사용할 수 있으나 test 데이터에 사용하지 말자

정규화 해제 : scaler.inverse_transform()

✿ StandardScaler

기본 스케일. 평균과 표준편차 사용

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(data)                # 모델 학습
```

✿ MinMaxScaler

최대/최소값이 각각 1, 0이 되도록 스케일링

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler( )
scaler.fit(data)
```

✿ MaxAbsScaler

최대절대값과 0이 각각 1, 0이 되도록 스케일링

```
from sklearn.preprocessing import MaxAbsScaler

scaler = MaxAbsScaler( )
scaler.fit(data)
```

✿ RobustScaler

중앙값(median)을 0으로 설정하고 IQR을 사용하여 아웃라이어의 영향을 최소화
사용 공식 ; (x−q2)/(q3−q1)

quantile_range 파라미터(default [0.25, 0.75])을 조정하여 더 넓거나 좁은 범위의 값을 이상치로 설정할 수 있음

```
from sklearn.preprocessing import RobustScaler

scaler = RobustScaler( )
scaler.fit(data)
```

3. 결측치 처리하기

✿ 결측치 인식

데이터의 요소가 결측치일 경우 TRUE 반환하고, 그렇지 않을 경우 FALSE 반환
변수별로 데이터의 각 행에 결측치가 있는지 확인 후 반환

```
dataframe.isna()
dataframe.isnull()
```

✿ 결측치가 있는/없는 행 확인

데이터의 요소가 결측치일 경우 TRUE 반환하고, 그렇지 않을 경우 FALSE 반환
변수별로 데이터의 각 행에 결측치가 있는지 확인 후 반환

```
# na가 있는 행들 저장
df_row_with_na = df [df.isna( ).any( axis = 1 )]

# na가 없는 행들 저장
df_row_without_na = df.dropna( )
```

✿ 결측치 제거

결측치가 존재하는 행/열 제거

데이터명.dropna()	행 기준으로 결측치가 있는 행은 모두 삭제
데이터명.dropna(axis=1)	열 기준으로 결측치가 있는 열은 모두 삭제
데이터명.dropna(how='all')	행 전체가 결측값인 행만 삭제
데이터명.dropna(thresh=2)	결측치의 갯수 수치를 지정해서 임계치를 넘어가면 삭제
데이터명.dropna(subset=['컬럼명1', '컬럼명2'])	특정 컬럼 내의 결측치만 삭제
데이터명.dropna(inplace=True)	결측치 제거한 상태를 바로 적용

✿ 결측치 대치법

해당 변수 값들을 특정 값으로 대치함

데이터명.fillna(0)	결측값을 0으로 대체
데이터명.fillna(method='ffill 또는 'pad')	결측값을 앞방향으로 채워나감
데이터명.fillna(method='bfill' 또는 'backfill')	결측값을 뒷 방향으로 채워나감
데이터명.fillna(method='ffill', limit=1)	앞/뒤 방향으로 결측값을 채우는 횟수를 1번으로 제한
데이터명.fillna(데이터명.mean())	결측값을 변수별 평균으로 대체
데이터명.dropna(inplace=True)	결측치 제거한 상태를 바로 적용
데이터명.fillna(데이터명.transform('median'))	결측치를 중앙값으로 대체
데이터명['컬럼명'].describe()['top']	범주형 변수 열의 최빈값으로 결측치를 대체

✿ KNN을 활용한 결측치 대치법

k-최근접 이웃을 사용하여 결측값 대치

```
imputer = KNNImputer(n_neighbors = int)          # 분석에 사용할 이웃의 수
df_filled = imputer.fit_transform(df)
df_filled = pd.DataFrame(df_filled, columns = df.columns)
```

4. 범주형 변수 변환하기

범주형 변수에 있는 원소를 columns로 변경
⇒ 해당 범주에 속하면 1, 아니면 0 으로 채우는 기법

✿ 범주형 변수 인코딩하기

```
pd.get_dummies( df ['범주형변수'] )
```

✿ 원본 데이터에서 범주형 변수를 더미변수 형태로 대체하기

```
pd.get_dummies( df, columns = [ '범주형1', '범주형2' ] )
```

5. 변수 축소(PCA)

1. 변수간의 스케일 차이가 주성분 선정에 영향을 주는 것을 방지하기 위해 정규화 수행 : StandardScaler()
2. 주성분분석 : sklearn의 PCA함수
3. 주성분의 설명력 확인 : pca.explained_variance_ratio_ 또는 Scree Plot

1. PCA를 위한 전처리

```
# 수치형 데이터만 추출
features = [ '수치형변수1', '수치형변수2' ]
x = df [ features ]

# 수치형 변수 정규화
from sklearn.preprocessing import StandardScaler
x = StandardScaler( ).fit_transform(x)
```

2. PCA

```
from sklearn.decomposition import PCA
pca = PCA( n_components = int )          # n_components : 생성할 주성분의 갯수
pca_fit = pca.fit(x)

print("고유 값 : ", pca.singular_values_ )
print("분산 설명력: ", pca.explained_variance_ratio_ )
```

singular_values_ : 전체 데이터에서 해당 모델(설정된 주성분의 개수)로 설명할 수 있는 분산의 비율

explained_variance_ratio_ : 전체 데이터에서 각 주성분이 설명할 수 있는 분산의 비율

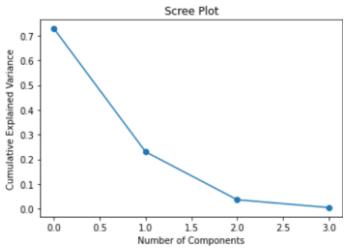
singular_values_와 **explained_variance_ratio_**로 사용할 주성분의 개수를 구할 수 있으나 Scree plot을 확인하여 정하기도 함

3.1. 사용할 주성분의 개수 정하기 : Scree Plot

해석 방법 : 플롯의 기울기(분산의 변화 정도)가 급격히 감소하는 지점의 바로 직전 지점까지를 주성분으로 선택한다

```
import matplotlib.pyplot as plt


plt.title( 'Scree Plot' )
plt.xlabel( 'Number of Components' )
plt.ylabel( 'Cumulative Explained Variance' )
plt.plot( pca.explained_variance_ratio_ , 'o-' )
```



3.2. 주성분으로 이루어진 새로운 데이터프레임 확인

```
pca = PCA(n_components = 2)                    # PCA 객체 생성 (주성분 갯수 2개 생성)
principalComponents = pca.fit_transform(x)     # 2개의 주성분을 가진 데이터로 변환
principalDf = pd.DataFrame( data = principalComponents, columns = [ 'pc1', 'pc2' ] )
principalDf
```

4. 주성분 산포도 확인

```
import matplotlib.pyplot as plt
import seaborn as sns


plt.title( '2 component PCA' )
sns.scatterplot( x = 'principal component1',
                 y = 'principal component2',
                 hue = Iris_data.target,
                 data = principalDf )
                # hue : 타겟 범주별로 다른 색상 지정
plt.show()
```

Part 3. EDA를 위한 그래프 그리기

✿ EDA에서 해야할 것

1. 분석 목적 확인 및 변수 이름과 타입 확인 : data.info()
2. 데이터 내용 확인 : head( ) 또는 tail( )로 데이터 내용 체크, pd.describe()
3. 개별 컬럼의 값 관찰
   a. 범위와 분포 : 산포도, 막대그래프, 히스토그램
   b. 이상치 확인 : 박스플롯 pd.boxplot()
4. 컬럼 간 상관관계 확인 : 히트맵

1. 산점도 (산포도)

두 개의 연속형 변수의 값에 따른 데이터의 분포를 확인하는 그래프

✿ 기본 scatter plot

```
import matplotlib.pyplot as plt


# 그래프 제목, x&y축 이름 지정 (옵션)
plt.title( '그래프 제목' )
plt.xlabel( 'Length' )
plt.ylabel( 'Weight' )

plt.scatter(x = df [ '컬럼명' ], y = df [ '컬럼명' ], alpha = 0.5)
plt.show()
```

✿ 타겟의 범주별 다른 색상/크기 옵션을 지정한 scatter plot

```
        import matplotlib.pyplot as plt
        import seaborn as sns

        plt.title( '2 component PCA' )
            sns.scatterplot( x = 'x축 컬럼명', y = 'y축 컬럼명', data = df,
              hue = 'target 컬럼명', style = 'target 컬럼명' )
plt.show()
```

2. 선 그래프

✿ 직선 그래프

y = ax+ b 형태의 직선이나 수평선/수직선을 그리는 그래프

```
        import matplotlib.pyplot as plt

            plt.plot( x축, 회귀식, data = df, c = 'color' )      # y = ax+ b 형태의
            직선
            plt.hlines( y, xmin , xmax )                          # y=h 형태의 수평선
            plt.vlines( x, ymin , ymax )                          # x=v 형태의 수직선
```

✿ 꺾은선 그래프

어떤 컬럼 값의 흐름에 따른 데이터의 변화를 파악하기 위해 사용
X축에 의해 정렬되어 있어야 함

```
        import matplotlib.pyplot as plt


            df = df.sort_values( by = [ 'x축' ] )
            plt.plot( 'x축', 'y축', data = df )

            # 특정 카테고리별로 그래프를 겹쳐 그릴 때 카테고리별로 plot을 그리고 범례를 제시함
            plt.plot( 'x축', 'y축', data = df .loc[df [ '범례' ] == '범례의 값' ])
            plt.legend( df.범례.unique( ) )
plt.show( )
```

✿ 곡선 그래프

곡선 모델을 만들고 x를 범위 내에서 균등한 간격으로 나누고
이에 대한 곡선모델의 값을 플로팅

```
        import numpy as np
        import matplotlib.pyplot as plt
        from scipy.interpolate import make_interp_spline


# make_interp_spline( ) : plot(x,y)의 값을 부드러운 곡선 형태로 만드는 모델 생성
model = make_interp_spline( df [ 'x축 컬럼명' ], df [ 'y축 컬럼명' ] )
x = np.linespace( n1, n2, n3 )        # n1부터 n2까지 n3등분한 값을 지정
            plt.plot( x, model( x ))
plt.show( )
```

예제 : 평균이 0이고 표준편차가 1인 확률밀도함수를 생성하고 그래프로 나타내기

```
        import numpy as np
        import matplotlib.pyplot as plt
        from scipy.stats import stats

x = np.linespace( n1, n2, n3 )      # n1부터 n2까지 n3등분한 값을 지정
y = stats.norm(0, 1).pdf(x)      # norm(0,1) : 평균 0, 표준편차 1인 정규 연속 확률변수
                                 pdf(x) : x에 대한 확률 밀도 함수
            plt.plot( x, y )
            plt.grid()           # 플롯에 격자 보이기
plt.show( )
```

3. 데이터의 건수 비교 : 막대 그래프, 히스토그램, 파이차트

막대그래프	범주형으로 구분된 변수의 범주별 데이터 수를 구한 것
	범주의 순서를 의도에 따라 변경 가능
히스토그램	연속된 수를 일정한 구간으로 나눈 후 구간별 데이터의분포를 그래프로 표현한 것
	임의로 범주의 순서를 바꿀 수 없음

✿ 수직 막대 그래프

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
data = df [ '변수명' ].value_counts()
        plt.bar( data.index, data.values,          # 데이터의 x, 높이
         width = 0.8, bottom = None,       # 막대의 넓이와 막대 밑면의 y좌표
         align = 'center', data = None      # 막대의 정렬, 데이터프레임
         color='black', edgecolor='black' ) # 그래프 색상, 그래프 테두리 색상
```

✿ 수평 막대 그래프

```
import pandas as pd
import matplotlib.pyplot as plt

data = df [ '변수명' ].value_counts()
plt.barh( data.index, data.values,
          width=0.8, bottom=None, align='center', data=None,
          color='black', edgecolor='black')
```

✿ 히스토그램

```
import pandas as pd
import matplotlib.pyplot as plt


          plt.hist( '변수명', bins = None, range = None, density = False, data = df )
          # bins : 히스토그램의 구간의 개수 정의
          # range : bin의 상한값과 하한값 (x.min(), x.max()) 형태로 선언
          # density : True이면 확률밀도함수를 그리고 반환
```

✿ boxplot

- 이상치를 탐지하기 위해서 사용
- 이상치가 있는지를 시각화 해주지만, 이상치의 index를 찾아 줄 때에는 IQR을 직접 적용하여 이상치가 있는 행들을 추출해야함

```
# 시각화
import pandas as pd
import matplotlib.pyplot as plt

        df.boxplot(column=None, by=None,)
        #  column : str or list of str, optional

        #  by : 범주형 변수가 있다면 범주형으로 나누어줄 수 있음 , seaborn 의
           hue와 비슷


# 이상치 index 추출 함수
def detect_outliers(df=None, column=None, weight=1.5):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)

    IQR = Q3 - Q1
    IQR_weight = IQR*weight

    outlier_idx = df[( df[column] < Q1 - IQR_weight ) | ( df[column] > Q3 +
                    IQR_weight )].index

    return outlier_idx
```

4. 상관관계 시각화

✿ 산점도 행렬

두 개 이상의 변수가 있는 데이터에서 변수들 간의 산점도를 그린 그래프

모든 변수에 대해서 두 변수의 산포도와 각 변수의 밀도 그래프(kde, histogram 등)을 그려 데이터의 분포와 변수들 간의 관계를 살펴볼 때 사용함

```
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix


# 단색 산점도 행렬 (종속변수를 함께 그릴 때 유용)
# 데이터프레임, 투명도(0 ~ 1), 그래프 크기(x, y), 대각선의 밀도 그래프 종류 (hist / kde)
scatter_matrix( df, alpha = 0.5, figsize = (8, 8), diagonal = 'hist' )
plt.show()
```

```
import matplotlib.pyplot as plt
import seaborn as sns


# 타겟의 범주별 색깔을 다르게 지정한 그래프 (종속변수가 범주형일 때 유용)
# 데이터프레임, 대각선 밀도 그래프 종류(auto, hist, kde), 색을 구분할 타겟 변수
sns.pairplot( df, diag_kind = 'auto', hue = 'target 컬럼명' )
plt.show()
```

해석 방법

1. 대각선의 히스토그램을 통해 이상치를 확인한다.
2. 종속변수와 설명변수들 간의 관계를 시각적으로 판단한다.
   종속변수가 수치형인 경우 각 설명변수와의 직선 상관관계를 비교한다.
   종속변수가 범주형인 경우 종속변수를 잘 구분하는 변수를 파악한다.
3. 설명변수 간의 직선 함수관계를 파악하여 다중공선성 문제를 진단한다.

✿ 히트맵

-1 ~ 1 사이의 값으로 두 변수 간의 관계의 정도를 비교하는 그래프

-1.0 < r ≤ -0.7	매우강한 음의 상관관계	-1.0 < r ≤ -0.7	매우강한 양의 상관관계
-0.7 < r ≤ -0.3	강한 음의 상관관계	0.7 < r ≤ 0.3	강한 양의 상관관계
-0.3 < r ≤- 0.1	약한 음의 상관관계	0.3 < r ≤ 0.1	약한 양의 상관관계
-0.1 < r ≤ 0.1	상관관계 없음		

```
import pandas as pd
import seaborn as sns

sns.heatmap(df,
            xticklabels = df.columns,
            yticklabels = df.columns,
            cmap='RdBu_r',
            annot=True,  # True: 상관계수 텍스트로 보이게 해줌
            linewidth=0.5)
```

5. Profiling

간단하게 데이터 EDA를 할 수 있는 도구

```
# jupyter에서아래 명령어 입력 (설치)
# !conda update -n base -c defaults conda -y
# !conda install -c conda-forge pandas-profiling=2.8 -y
# !conda install -c conda-forge ipywidgetsy -y
# !conda install pandas=1.2 -y

from pandas_profiling import ProfileReport
ProfileReport(df)
```