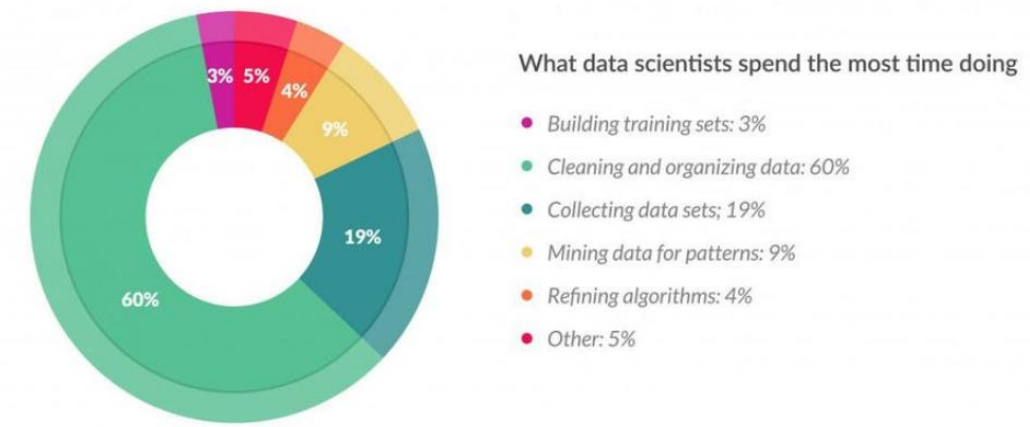
11. 데이터 전처리

데이터 전처리란 ?

- 데이터 분석 결과에 직접적인 영향을 미치는 과정
- 데이터 분석에서 가장 많은 시간이 소요됨
  - 데이터 정제 (중복 값, 결측 값, 이상 값 처리)
- 분석 변수 선택, 추가, 변환
  - 분석에 관련성이 높은 변수 선정
  - 파생 변수 정의
  - 변수 변환 (encoding, binning, normalization)



source : Forbes

중복 데이터 처리

- 중복 데이터 : 특정 열 또는 모든 열에 대해 동일한 데이터 값을 갖는 경우
  - 데이터를 수집 및 병합 하는 과정에서 오류로 데이터가 중복되는 경우가 발생할 수 있음

- 판다스 중복 데이터 처리 관련 주요 메소드
  - `df.duplicated()`: 중복 값이 있는지 여부 확인
  - `df.drop_duplicates()`: unique 한 1개의 행만 남기고 나머지 제거
  - `df.reset_index()` : 필요시 행 인덱스 초기화

중복 데이터 처리

```
dict_1 = {
    'item_id': [1, 1, 2, 2, 2, 2, 3, 4],
    'company': ['A', 'A', 'B', 'B', 'B', 'C', 'A', 'C'],
    'item_name': ['apple', 'apple', 'banana', 'banana', 'banana', 'banana', 'grape', 'watermelon'],
    'price': [15000, 15000, 4750, 4750, 4750, 4200, 13000, 18000],
    'color': ['red', 'red', 'yellow', 'yellow', 'yellow', 'yellow', 'purple', 'green']
}

df_1 = pd.DataFrame(dict_1)

print(df_1)
```

	item_id	company	item_name	price	color
0	1	A	apple	15000	red
1	1	A	apple	15000	red
2	2	B	banana	4750	yellow
3	2	B	banana	4750	yellow
4	2	B	banana	4750	yellow
5	2	C	banana	4200	yellow
6	3	A	grape	13000	purple
7	4	C	watermelon	18000	green

중복 데이터 처리

```
duplicated_bool = df_1.duplicated()
print(duplicated_bool)    # (1)
df_1.drop_duplicates(inplace=True)
print(df_1)               # (2)
print( df_1.loc[6] )      # (3)
print( df_1.iloc[3] )     # (3)
```

```
0    False
1     True
2    False
3     True
4     True
5    False
6    False
7    False
dtype: bool
```

(1)

	item_id	company	item_name	price	color
0	1	A	apple	15000	red
2	2	B	banana	4750	yellow
6	3	A	grape	13000	purple
7	4	C	watermelon	18000	green

(2)

```
item_id          3
company          A
item_name    grape
price        13000
color       purple
Name: 6, dtype: object

item_id          3
company          A
item_name    grape
price        13000
color       purple
Name: 6, dtype: object
```

(3)

중복 데이터 처리 후 인덱스 초기화

- 중복 데이터 drop 이후 행 인덱스는 원본 인덱스 값으로 유지 되어 있음
  - .loc 및 .iloc 속성을 상황에 맞게 사용해야 함
  - reset_index() 메소드로 인덱스 초기화 가능

```
print(df_1)
print()
df_1.reset_index( inplace=True, drop=True )
print(df_1)
```

	item_id	company	item_name	price	color
0	1	A	apple	15000	red
2	2	B	banana	4750	yellow
5	2	C	banana	4200	yellow
6	3	A	grape	13000	purple
7	4	C	watermelon	18000	green

	item_id	company	item_name	price	color
0	1	A	apple	15000	red
1	2	B	banana	4750	yellow
2	2	C	banana	4200	yellow
3	3	A	grape	13000	purple
4	4	C	watermelon	18000	green

결측 데이터 처리

- 결측 값 : 값이 비어 있는 데이터
  - 판다스에선 주로 NaN (Nat a Number) 로 표현
- 결측 값 유형
  - 사람에 의한 오류 (미입력)
    - 개인 정보 (예: 연락처)
    - 관련이 없다고 판단한 항목 (예: 은퇴한 경우 소속 회사 공란)
  - 전산 오류, 장비 오류, 전송 오류

결측 데이터 확인

- **pd.isna() or pd.isnull()** : 시리즈나 데이터프레임을 받아서 NaN 위치를 확인

```
import pandas as pd
import numpy as np

dict_1 = {
    'col1': [1, 2, 3, np.nan, np.nan],
    'col2': [10, 20, np.nan, 40, 50],
    'col3': [np.nan, 200, 300, 400, np.nan]
}
df_1 = pd.DataFrame(dict_1)
print(df_1)
print()
print( pd.isna(df_1) )
#print(pd.isnull(df_1))
```

```
   col1  col2   col3
0   1.0  10.0    NaN
1   2.0  20.0  200.0
2   3.0   NaN  300.0
3   NaN  40.0  400.0
4   NaN  50.0    NaN

    col1   col2   col3
0  False  False   True
1  False  False  False
2  False   True  False
3   True  False  False
4   True  False   True
```

결측 데이터 확인

- sum( ) 함수를 사용하여 열의 합계 확인
  - 열 별로 결측값 개수 및 데이터프레임 전체 결측값 개수 확인
- count( ) 메소드로 결측이 아닌 항목 개수 확인

```
print( pd.isna(df_1).sum() )
print( pd.isna(df_1).sum().sum() )
print()
print( df_1.count() )
print( df_1.count().sum() )
print()
print( df_1.shape )
print( df_1.shape[0] * df_1.shape[1] )
```

```
col1    2
col2    1
col3    2
dtype: int64
5

col1    3
col2    4
col3    3
dtype: int64
10

(5, 3)
15
```

결측 데이터 처리 방법

- 결측 값이 있는 행 (혹은 열) 제거
  - df.dropna( ) : 결측 값이 있는 행을 삭제
  - df.dropna(axis=1) : 결측 값 있는 열을 삭제

가장 간편함

그러나 너무 많은 데이터를 잃어버릴 수 있음

- 결측 값을 특정 값으로 채우기
  - df.fillna(특정 값) : 결측 값을 특정 값으로 채우기
  - 예: 0, 평균 값, 혹은 보간 법으로 채우기

해당 분야의 전문가와 논의하여

적절한 값을 설계하는 것이 중요

결측 데이터 처리 방법

```
df_2 = df_1.copy()
# 결측 값 존재하는 행 제거
df_2.dropna( inplace=True )
print( df_2 )
```

```
   col1  col2   col3
1   2.0  20.0  200.0
```

```
df_2 = df_1.copy()
# 결측 값을 0으로 채우기
df_2.fillna( 0, inplace=True )
print( df_2 )
```

```
   col1  col2   col3
0   1.0  10.0    0.0
1   2.0  20.0  200.0
2   3.0   0.0  300.0
3   0.0  40.0  400.0
4   0.0  50.0    0.0
```

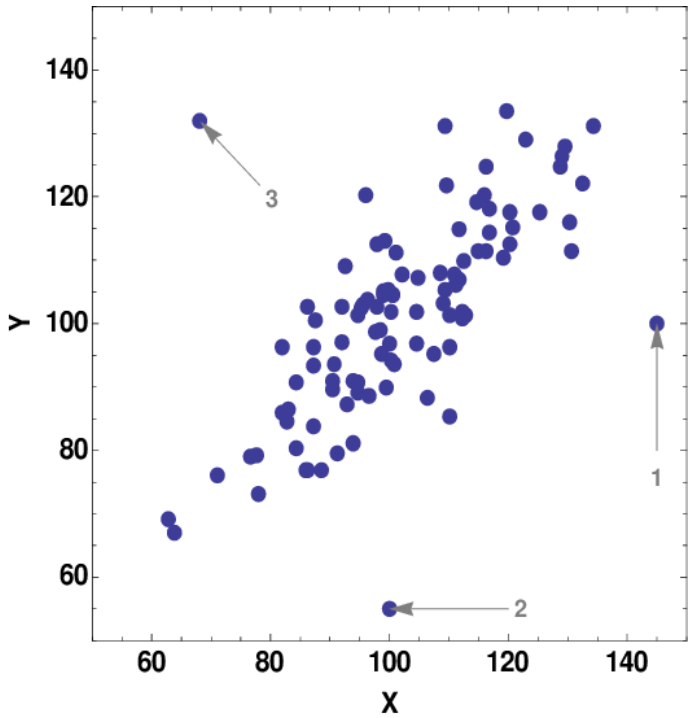
```
df_2 = df_1.copy()
# 각 열 별로 평균 값 구하기
print( df_2.mean() )
print()
# 결측 값을 각 열 평균 값으로 채우기
df_2.fillna( df_2.mean(), inplace=True )
print(df_2)
```

```
col1      2.0
col2     30.0
col3    300.0
dtype: float64

   col1  col2   col3
0   1.0  10.0  300.0
1   2.0  20.0  200.0
2   3.0  30.0  300.0
3   2.0  40.0  400.0
4   2.0  50.0  300.0
```

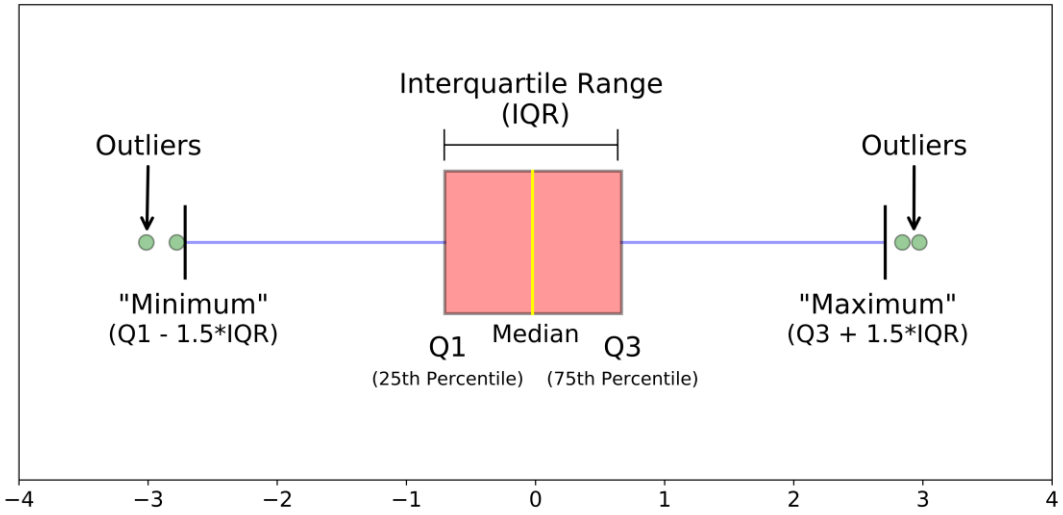
이상 (outlier) 데이터 처리

- 이상 데이터란 다른 데이터보다 아주 작은 값이나 아주 큰 값
- 데이터를 분석할 때 이상치는 분석 및 의사결정에 영향을 미침
- 이상치 확인 방법
  - 사분위수 기반 검출
- 이상치 처리 방법
  - 제거
  - 치환 (평균, 최소, 최대 값 등으로 치환)



이상 데이터 검출

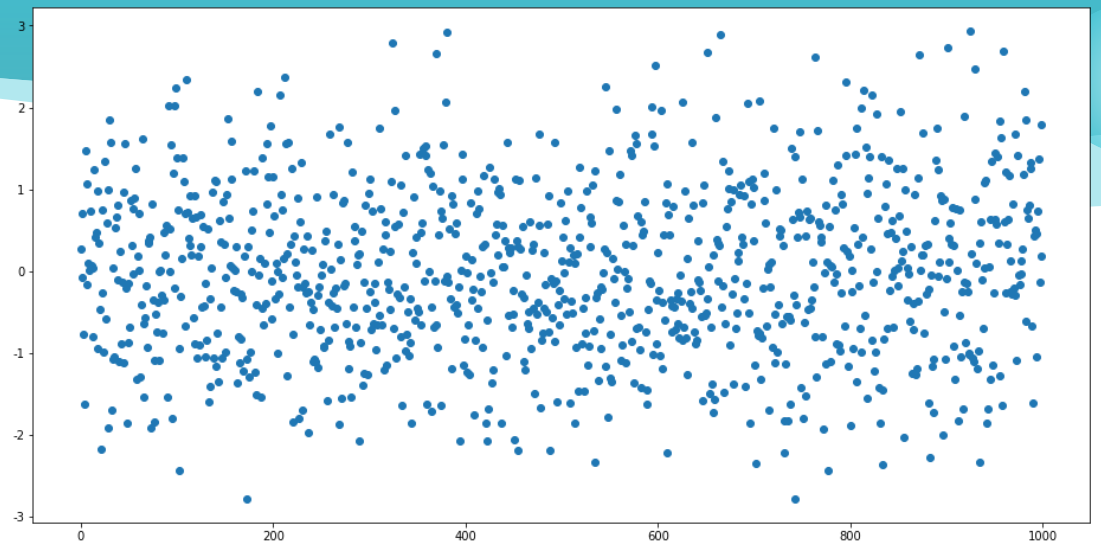
사분위수	설명
제1 사분위수(Q1)	• 데이터의 25%가 이 값보다 작거나 같음
제 2 사분위수(Q2)	• 중위수 데이터의 50%가 이 값보다 작거나 같음
제3 사분위수(Q3)	• 데이터의 75%가 이 값보다 작거나 같음
사분위간 범위(IQR)	• 제1 사분위수와 제3 사분위수 간의 거리 (Q3-Q1) • 데이터의 중간 50% 데이터



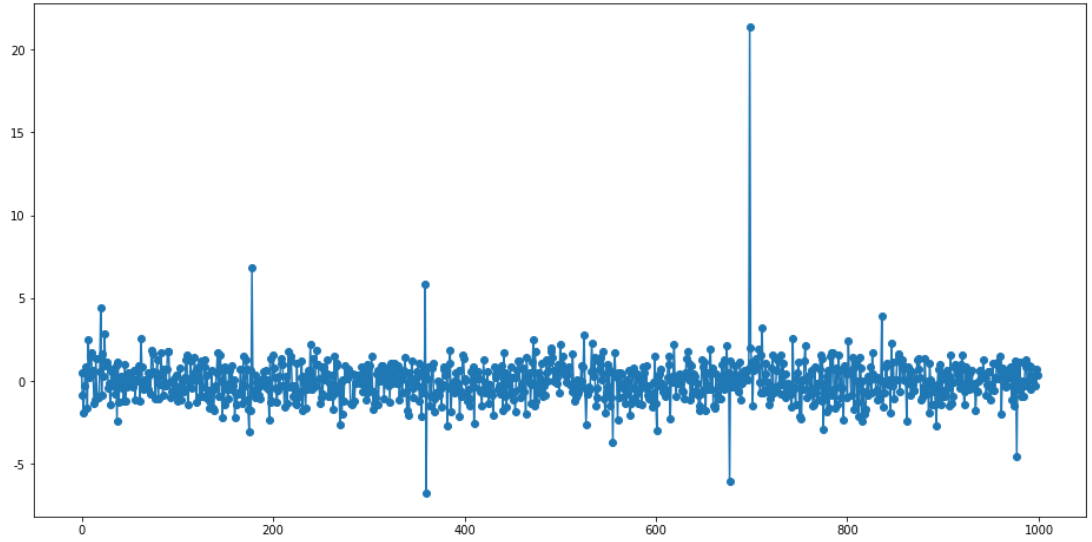
- **낮은 이상치** : (Q1 - 1.5 * IQR) **보다 작은 값**
- **높은 이상치** : (Q3 + 1.5 * IQR) **보다 높은 값**

이상 데이터 검출

```
data = np.random.randn(1000)
plt.figure(figsize=(16,8))
plt.plot(data, 'o')
plt.show()
```



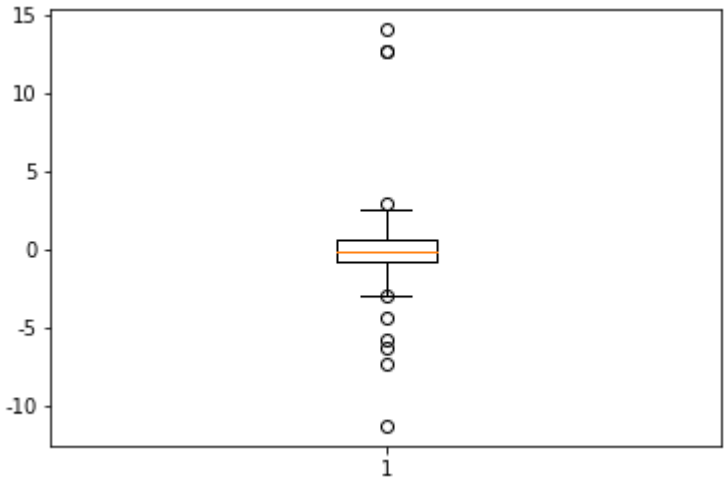

```
ol_data = data.copy()
ol_num = 10
for i in range(ol_num):
    rand_id = np.random.randint(0, len(data))
    ol_data[rand_id] = ol_data[rand_id] * 5
plt.figure(figsize=(16,8))
plt.plot(ol_data, 'o-')
plt.show()
```



이상 데이터 검출

- 사분위수 메소드 quantile( ) 사용

```
plt.boxplot(ol_data)
Q1 = ol_data.quantile(0.25)
Q3 = ol_data.quantile(0.75)
IQR = Q3-Q1
print("Q1:", Q1, "Q3:", Q3, "IQR:", IQR)
print( ol_data[ ol_data > (Q3 + 1.5 * IQR) ].count() )
print( ol_data[ ol_data < (Q1 - 1.5 * IQR) ].count() )
```

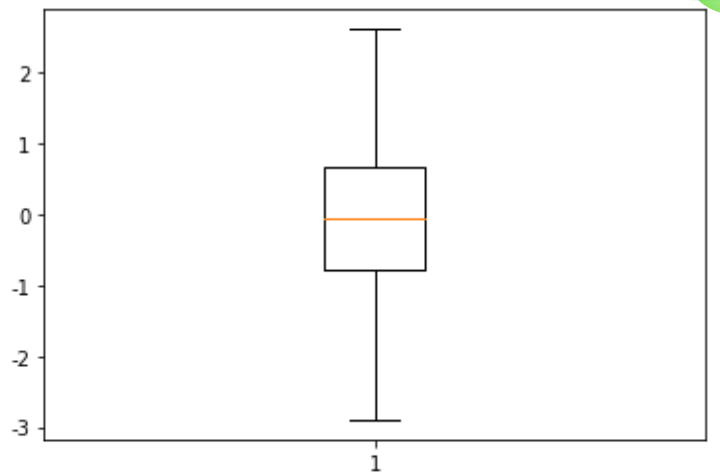


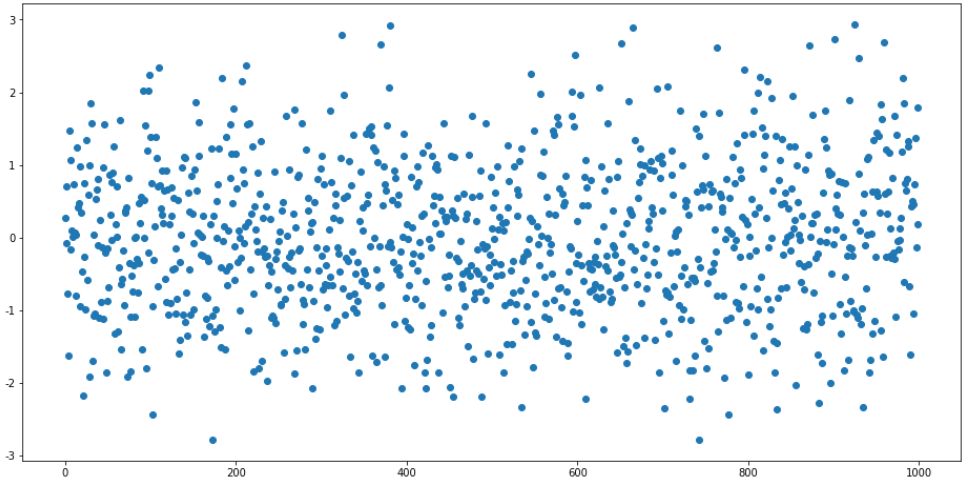

```
Q1 : -0.7750709600209859
Q3 : 0.6389728441539163
IQR: 1.4265718023648066
4
6
```

이상 데이터 삭제

```
idx1 = ol_data[ ol_data > (Q3 + 1.5 * IQR)].index
idx2= ol_data[ ol_data < (Q1 - 1.5 * IQR)].index
ol_data.drop( idx1, inplace=True )
ol_data.drop( idx2, inplace=True )
print( ol_data.count() )
plt.boxplot( ol_data )
plt.figure(figsize=(16,8))
plt.plot(ol_data, 'o-')
plt.show()
```

```
990
```





데이터 변환 : apply 메소드

- 정의해 놓은 함수를 데이터프레임 또는 특정 열에 일괄적으로 적용가능

```
def process(x):
  if x >= 90:
    return 'A'
  elif x >= 80:
    return 'B'
  else:
    return 'C'
```

```
names = [ '춘향', '몽룡', '향단', '방자']
math = [90, 85, 100, 95]
df = pd.DataFrame( {'이름':names, '수학':math } )
print(df)
print()
tmp = df.copy()
tmp['수학'] = tmp['수학'].apply( process )
print(tmp)
print()
df['학점'] = df['수학'].apply( process )
print(df)
```

```
   이름   수학
0  춘향   90
1  몽룡   85
2  향단  100
3  방자   95

   이름 수학
0  춘향  A
1  몽룡  B
2  향단  A
3  방자  A

   이름   수학 학점
0  춘향   90  A
1  몽룡   85  B
2  향단  100  A
3  방자   95  A
```

데이터 전처리 연습 : 타이타닉호 생존자 데이터셋

- 북대서양 횡단 여객선
  - 당시 세계에서 가장 큰 배
  - 영국에서 뉴욕으로 첫 항해
- 빙산과 충돌 사건 (1912년 4월 14일)
  - 탑승인원 2,224명
  - 구조자 710명 (사망자 1,514명)

seaborn 타이타닉 데이터셋

```
import seaborn as sns
titanic = sns.load_dataset('titanic')
print(titanic)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
..	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

[891 rows x 15 columns]

타이타닉 데이터셋 설명

컬럼 명	의미	데이터 값
survived	생존여부	0 (사망) / 1 (생존)
pclass	객실 등급 (숫자)	1 / 2 / 3
sex	성별	male/female
age	나이	0 ~ 80
sibsp	형제자매 / 배우자 인원수	0 ~ 8
parch:	부모 / 자식 인원수	0 ~ 6
fare:	요금	0 ~ 512.3292
embarked	탑승 항구	S (Southampton) C (Cherbourg) Q (Queenstown)
class	좌석등급 (영문)	First, Second, Third
who	성별	man / woman
deck	객실 고유 번호 가장 앞자리 알파벳	A,B,C,D,E,F,G
embark_town	탑승 항구 (영문)	Southampton / Cherbourg / Queenstown
alive	생존여부 (영문)	no (사망) / yes (생존)
alone	혼자인지 여부	True (가족 X) / False (가족 O)

데이터 전처리 : 원본 데이터 copy (백업)

```
view = titanic          # 데이터 연결하기
print( id(view), id(titanic) )
# 두 dataframe 변수가 동일한 메모리 주소
# 한 dataframe 을 변경하면 동일하게 적용 됨
print()
data = titanic.copy() # 데이터 복사하기
print( id(data), id(titanic) )
# 각 dataframe 변수는 서로 다른 메모리 주소
# 한 dataframe 을 변경해도 서로 영향을 주지 않음
print()
print(data.count())
```

```
139788958280208 139788958280208

139788976390992 139788958280208

survived       891
pclass         891
sex            891
age            714
sibsp          891
parch          891
fare           891
embarked       889
class          891
who            891
adult_male     891
deck           203
embark_town    889
alive          891
alone          891
dtype: int64
```

데이터 전처리 : 중복 데이터 처리

```
dup = data.duplicated()
print(dup.sum()) # 중복 데이터 개수 출력
print()
# 중복 데이터 삭제
data.drop_duplicates(inplace=True)
print(data.count())
print()
print(titanic.count())
```

```
107

survived      784
pclass        784
sex           784
age           678
sibsp         784
parch         784
fare          784
embarked      782
class         784
who           784
adult_male    784
deck          202
embark_town   782
alive         784
alone         784
dtype: int64
```

data

```
survived      891
pclass        891
sex           891
age           714
sibsp         891
parch         891
fare          891
embarked      889
class         891
who           891
adult_male    891
deck          203
embark_town   889
alive         891
alone         891
dtype: int64
```

titanic

데이터 전처리 : 결측 데이터 처리

```
# 일부 열만 가져오기
data2 = data[ ["survived", "sex", "fare", "age" ] ]
# 각 열 데이터 개수 확인
print(data2.count())
print()
# 결측 데이터 개수 확인
print(data2.isna().sum())
print()
# 결측 데이터 삭제
data2.dropna( inplace=True )
# 각 열 데이터 개수 확인
print(data2.count())
```

```
survived    784
sex         784
fare        784
age         678
dtype: int64

survived      0
sex           0
fare          0
age         106
dtype: int64

survived    678
sex         678
fare        678
age         678
dtype: int64
```

데이터 전처리 : 이상 데이터 처리

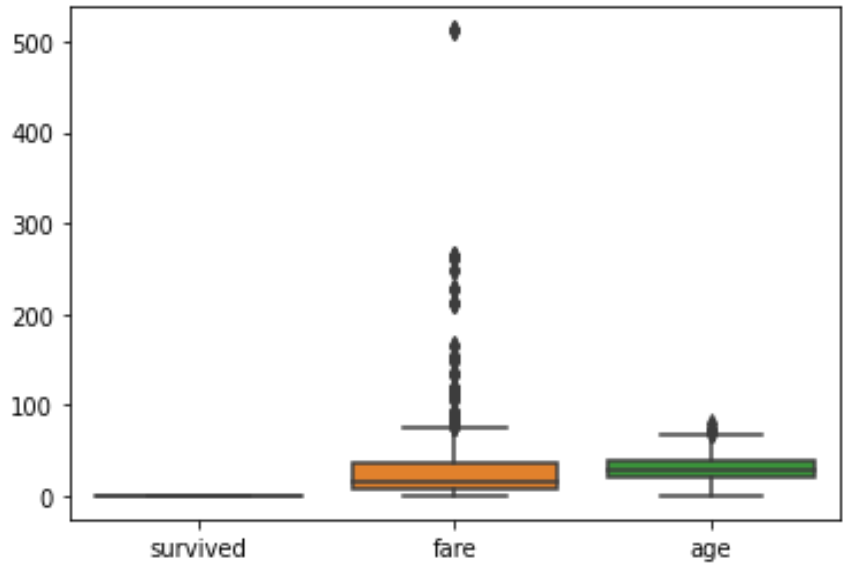
사분위 기준 이상 데이터 존재 유무 확인

해당 데이터를 확인하여 이상 여부를 점검

```
sns.boxplot(data=data2)
plt.show()
```

```
ser = data2['fare']
Q1 = ser.quantile(0.25)
Q3 = ser.quantile(0.75)
print(Q1, Q3)
IQR = Q3-Q1
idx =  ser[ ser > (Q3 + 1.5 * IQR)].index
print( ser[idx].sort_values(ascending=False) )
```

```
8.05 35.28855
679    512.3292
737    512.3292
258    512.3292
27     263.0000
438    263.0000
         ...
102     77.2875
681     76.7292
52      76.7292
645     76.7292
218     76.2917
Name: fare, Length: 88, dtype: float64
```



데이터 전처리 : 이상 데이터 처리

- 범주형 데이터 이상 여부는 ?
  - 유효한 값들만 존재하는지 확인

```
print( data2['survived'].value_counts() )
print()
print( data2['sex'].value_counts() )
```

```
0    394
1    284
Name: survived, dtype: int64

male      422
female    256
Name: sex, dtype: int64
```

데이터 전처리 : 데이터 변환

- 데이터 분석, 학습 을 위해 범주형/문자열 데이터를 숫자 (코드) 로 변환

```
def encoding(x):
  if x == 'male':
    return 0
  else:
    return 1
```

```
data2['sex_code'] = data2['sex'].apply(encoding)
print( data2['sex'].value_counts() )
print()
print( data2['sex_code'].value_counts() )
```

```
male      422
female    256
Name: sex, dtype: int64

0    422
1    256
Name: sex_code, dtype: int64
```

타이타닉 데이터셋 전처리 완성

```
print(titanic)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
..	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

[891 rows x 15 columns]

```
print(data2)
```

	survived	sex	fare	age	sex_code
0	0	male	7.2500	22.0	0
1	1	female	71.2833	38.0	1
2	1	female	7.9250	26.0	1
3	1	female	53.1000	35.0	1
4	0	male	8.0500	35.0	0
..	...	...	...	...	...
883	0	male	10.5000	28.0	0
885	0	female	29.1250	39.0	1
887	1	female	30.0000	19.0	1
889	1	male	30.0000	26.0	0
890	0	male	7.7500	32.0	0

[678 rows x 5 columns]

```
data2.reset_index(drop=True, inplace=True)
print(data2)
```

	survived	sex	fare	age	sex_code
0	0	male	7.2500	22.0	0
1	1	female	71.2833	38.0	1
2	1	female	7.9250	26.0	1
3	1	female	53.1000	35.0	1
4	0	male	8.0500	35.0	0
..	...	...	...	...	...
673	0	male	10.5000	28.0	0
674	0	female	29.1250	39.0	1
675	1	female	30.0000	19.0	1
676	1	male	30.0000	26.0	0
677	0	male	7.7500	32.0	0

[678 rows x 5 columns]